Design and Evaluation of a Hybrid Multi-Task Learning Model for Optimizing Deep Reinforcement Learning Agents

by

Nelson Vithayathil Varghese

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Department of Electrical, Computer and Software Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April 2021

THESIS EXAMINATION INFORMATION

Submitted by: **Nelson Vithayathil Varghese**

Master of Applied Science in **Electrical and Computer Engineering**

Thesis title: Design and Evaluation of a Hybrid Multi-Task Learning Model for Optimizing Deep Reinforcement Learning Agents

An oral defense of this thesis took place on March 22, 2021, in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Ying Wang
Research Supervisor	Dr. Qusay H. Mahmoud
Examining Committee Member	Dr. Akramul Azim
Thesis Examiner	Dr. Kourosh Davoudi

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Design and Evaluation of a Hybrid Multi-Task Learning for Optimizing the Deep Reinforcement Learning Agents

Nelson Vithayathil Varghese — Advisor:

Ontario Tech University, 2021 — Dr. Qusay H. Mahmoud

Driven by recent technological advancements within the artificial intelligence domain, deep learning has emerged as a promising representation learning technique. This in turn has given rise to the evolution of deep reinforcement learning that combines deep learning with reinforcement learning methods. Subsequently, performance optimization achieved by reinforcement learning intelligent agents designed with model-free based approaches were predominantly limited to systems with reinforcement learning algorithms learning single task. Such a model was found to be quite data inefficient, whenever agents needed to interact with more complex, rich data environments. This thesis introduces a hybrid multi-task learning-oriented approach for optimization of deep reinforcement learning agents operating within different but semantically similar environments with related tasks. Empirical results obtained with OpenAI Gym library-based Atari 2600 video gaming environment demonstrate that the proposed hybrid multi-task learning model is successful in addressing key challenges associated with the performance optimization of deep reinforcement learning agents.

Keywords: Deep Reinforcement Learning; Neural Networks; Deep Learning; Multi-task Learning; Actor-Critic

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Nelson Vithayathil Varghese

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis, and I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

Results from this thesis research have been disseminated in the following publications:

- N. Vithayathil Varghese and Q. H. Mahmoud, "A Hybrid Multi-Task Learning Approach for Optimizing Deep Reinforcement Learning Agents," *IEEE Access, vol. 9, pp. 44681--44703, 2021 (23 pages); DOI: 10.1109/ACCESS.2021.3065710*

- N. Vithayathil Varghese and Q. H. Mahmoud, "Optimization of Deep Reinforcement Learning with Hybrid Multi-Task Learning," in *IEEE International Systems Conference (SysCon)*, Vancouver, Canada, 2021. *(Accepted, 8 pages)*

- Vithayathil Varghese, N. and Mahmoud, Q.H., 2020. A survey of multi-task deep reinforcement learning. *Electronics 2020, 9(9), 1363 (21 pages); https://doi.org/10.3390/electronics9091363*.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Almighty God for giving me such an opportunity to work in this research field. After that, I would like to express my sincere gratitude to my respected thesis supervisor Dr. Qusay H. Mahmoud for his generous support and guidance throughout my graduate studies. He provided me continuous guidelines and suggestions all the time that helped me to grow as an independent researcher. I would also like to extend my gratitude to all of the course instructors for their valuable guidance in every step of my learning stage during the graduation period. Finally, I would like to thank my parents, family members, and Ontario Tech University for their encouragement and valuable support that has helped me to complete the research successfully.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 2

CHAPTER 4

CHAPTER 5

LIST OF FIGURES

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

LIST OF ABBREVIATIONS AND SYMBOLS

AI – Artificial Intelligence

ML – Machine Learning

DL - Deep Learning

RL - Reinforcement Learning

DRL - Deep Reinforcement Learning

A3C - Asynchronous Advantage Actor-Critic

PD - Policy Distillation

AM - Actor Mimic

AGI - Artificial General Intelligence

MDP - Markov Decision Process

MNIST - Modified National Institute of Standards and Technology

CIFAR - Canadian Institute for Advanced Research

DQN - Deep Q-Network

ALE - Arcade Learning Environment

DoF - Degrees of Freedom

IMPALA - Importance Weighted Actor-Learner Architecture

DISTRAL - DIStil and TRAnsfer Learning

KL - Kullback-Leibler

RNN – Recurrent Neural Network

LSTM - Long Short-Term Memory

CPU - Central Processing Unit

PG - Policy Gradient

CNN - Convolutional Neural Network

GPU - Graphics Processing Unit

AC - Actor-Critic

CUDA - Compute Unified Device Architecture

HPC – High-Performance Computing

MTDRL - Multi-Task Deep Reinforcement Learning

RAM - Random-Access Memory

S - state space

A - action space

r - the reward at any time step

S_t - state at time t

S_{t+1} - state at time $t+1$

a – action at any time step

π - policy

π^*- optimal policy

$\pi(a|s)$ – stochastic policy

A_t - action at time t

$\mathscr{R}$ - reward space

R_t - reward at time t

E - environment

γ - discount factor

$V(s)$ - state-value function

$Q(s, a)$ - action-value function

$A(s, a)$ - advantage function

α - learning rate

$P(s`, r|s, a)$ – transition probability

s, a, s`, r - state, action, next state, reward tuple

$V^{\pi}(s)$ – value of a state with policy π

$Q^{\pi}(s, a)$ - value of (state, action) pair with policy π

Chapter 1

Introduction

Over the last few decades, the reinforcement learning domain has been well established its position as a vital topic within technological areas such as robotics and intelligent agents [1]. The core objective of RL is to address the problem of how reinforcement learning agents should explore their environment optimally, and thereby learn to take optimal actions to achieve the highest possible reward while in a given state [2]. Supported by recent advancements within the machine learning field, RL has been cemented its position as one of the major machine learning paradigms that deal with an RL agent's behavior pattern within an environment. In comparison to the performance of machine learning systems based out of contexts such as supervised learning, and unsupervised learning, oftentimes performance of traditional RL agents was not optimal. This was primarily due to the difficulties related to deducing the optimum policy out of the massive state-action space associated with the environment of RL problems. At the same time, the inception of deep learning with its very high level of representational learning capability has given a new dimension to the field of reinforcement learning namely, deep reinforcement learning. As a result of these advancements, DRL agents have been applied to various areas such as continuous action control,3D first-person environments, and gaming. Especially in the field of gaming, DRL agents are proven to be extremely successful and could surpass the human-level performance on classic video games like Atari as well as board games such as chess and Go [3].

Despite the impressive results with a single-task-based approach, the RL agent is found to be less efficient with environments that are more complex and richer in data such as 3D environments. One of the directions to improve the efficiency of the RL agent in such an environment is by the application of multi-task-based learning. During multi-task learning, a set of closely related tasks from the operating environment will be learned concurrently by individual agents with the help of a deep reinforcement algorithm such as A3C (Asynchronous Advantage Actor-Critic) [4]. With this approach, at regular intervals, the neural network parameters of each of these individual agents will be shared with a global network. By combining the learning parameters of all the individual agents, the global network derives a new set of parameters, which will be shared back with all the agents. The key objective of this methodology is to enhance the overall performance of the RL agent by transferring the learning, shared knowledge, among multiple related tasks running within the same environment. One of the most widely accepted multi-task learning methodologies within reinforcement learning is named parallel-based multi-task learning, in which a single RL agent master a group of diverse tasks [5]. The core idea behind this approach mainly relies on the architecture used by the deep reinforcement learning model based on a single learner, often known as a critic, combined with different actors. Each of the individual actors generates their learning trajectories, which are a set of parameters, and sends them to the learner module, also called a critic module either synchronously or asynchronously fashion. After this stage, each of the actors retrieves the latest set of policy parameters from the learner before the next learning trajectory begins. With this approach, learnings from each of the individual tasks will be shared with every other task, which internally improves the overall learning momentum of the RL agent.

The major motivation behind the proposed hybrid multi-task approach is to address some of the major challenges associated with the existing multi-task deep reinforcement learning (MTDRL) paradigm. Especially, attempting to address key challenges such as partial observability, effective exploration, and lastly the amount of training time required to achieve an acceptable level of performance.

1.1 Contributions

The main contribution of this thesis is a hybrid multi-task learning model for the optimization of the performance of deep reinforcement learning agents. The vision of this thesis is that the hybrid multi-task model designed, implemented, and evaluated will serve as a prototype for addressing the challenges mentioned in the previous section. To this end, the following research contributions are presented:

- Design and development of a hybrid multi-task learning model to optimize the performance of DRL agents.
- Evaluation of DRL agent's performance with hybrid multi-task learning model within the context of the aforementioned challenges.
- Empirical analysis of the optimization in DRL agent's performance with hybrid multi-task learning model by using the OpenAI Gym library-based Atari 2600 game environments. The analysis is conducted with multiple games having both a high level of similarity and dissimilarity.

1.2 Thesis Outline

This remainder of this thesis is structured as follows.

Chapter 2 presents the background of reinforcement learning which includes its ecosystem, Markov property, and key challenges associated with reinforcement learning. Further on, this chapter explains the various existing approaches that are attempted on the multi-tasking front of DRL. Finally, it also covers the details on various related work done within the same arena, with special focus given on three of the state-of-the-art frameworks namely DISTRAL, IMPALA, and PopArt.

Chapter 3 provides details on the design aspects of the proposed hybrid multi-task learning model and its architecture. To this end, this chapter starts by introducing the details of actor-critic learning methodology, followed by the information on the role of both actor and critic in the overall learning process. Subsequently, this chapter explains how to leverage the actor-critic model's parallel, multi-task learning capabilities into the proposed hybrid multi-task learning model by adopting the A3C into multi- gaming(hybrid) environment.

Chapter 4 includes the details on the implementation of a hybrid multi-task learning model by using the worker agents. To this end, this chapter provides various aspects related to the worker agents, such as the architecture of worker agents, the training workflow of worker agents, and how worker agents could be extended to a multi-task learning environment. Along with this, it also covers the specific details of worker agent’s implementation by using neural networks such as CNN and how it will be applied to the OpenAI Gym library. Additionally, this chapter also gives details on the various machine learning libraries and IDE used for the implementation.

Chapter 5 covers the information related to all the experiments conducted and related results obtained with the hybrid multi-task model. The first section of this chapter focuses on the evaluation of the hybrid multi-task model by using the Atari 2600 based gaming environments and various test configurations attempted. Details related to the experiments conducted with both desktop-based test environment as well as cloud-based environment namely Paperspace are provided in this part. Further on, the second section of the chapter focuses on analyzing both the test results obtained as well as how well a hybrid multi-task learning model could optimize the performance of DRL agents by mitigating the DRL challenges.

Finally, **Chapter 6** provides the conclusion and details of the future work planned.

1.3 Summary

This chapter provided a preliminary discussion on some of the key challenges associated the multi-task deep reinforcement learning. To this end, this chapter has introduced the core goal of this thesis as to design a hybrid multi-task learning model to optimize the performance of the deep reinforcement learning agent. Chapter 2 discusses the background and eco-system of reinforcement learning, along with the details of various multi-task learning approaches on the deep reinforcement learning front and related works done. The main contribution of this thesis is a hybrid multi-task learning model for the optimization of the performance of deep reinforcement learning agents.

Chapter 2

Background and Related Work

This chapter provides information on reinforcement learning background by including the details such as RL ecosystem, Markov property, and key challenges associated with RL-based intelligent agents. In addition to this, this chapter also presents the topic of multi-task learning, followed by various approaches and techniques that are developed for achieving multi-task learning on the RL front. Following this, details regarding the related work carried out on the multi-task learning on the DRL front is presented, with special focus given on three of the DRL multi-task learning state-of-the-art frameworks namely Distal, IMPALA and PopArt.

2.1 Reinforcement Learning

Reinforcement learning is one of the prominent ML paradigms dealing with sequential decision-making that involves mapping situations to actions in a way that maximizes the associated reward. Within RL ecosystems, the learner, which is also known as an agent, is not explicitly instructed on which actions to take at each timestep *t*, but instead, the RL agent must follow a trial-and-error method to identify which actions generate the most reward. One of the most challenging aspects of the RL is that actions that have already been carried out may affect not only the immediate reward but also the further states and, through that, all subsequent rewards. Reinforcement learning distinguishes itself from other machine learning methods by the above two characteristics--trial-and-error search and delayed reward [1].

2.1.1 The Ecosystem of Reinforcement Learning

A standard reinforcement learning setup consists of an agent situated within an environment *E*, where an agent will be interacting with the environment in discrete timesteps. At each of these timesteps *t*, the agent will be in a state S_t $(S_t \in S)$ and will be performing a chosen action A_t $(A_t \in A)$ within the environment *E*. Further on, the environment responds by updating the current state S_t to a follow-up state S_{t+1} with a new timestep $t+1$ and also gives a reward $r(S_t, A_t) \in \mathscr{R}$ to the agent, indicating the reward value of performing an action in the preceding state *St* [1]. Fig. 2.1 below represents the standard ecosystem for a reinforcement learning environment at any given timestep t. By performing multiple actions in a sequential learning manner in a sequence of associated states *s*, with related actions a, respective follow-up states *s'* and rewards *r*, several episodes of tuples of <s, a, s', r> are generated. At any given state S_t, the goal of the agent is to determine a policy π that can create a state-to-action mapping that maximizes the accumulated reward over the lifetime of the agent for that particular state [6].

Figure 2.1: The ecosystem of Reinforcement Learning.

At any point in the time *t*, the goal of the RL agent is to select the actions in such a way that it maximizes its expected return. The reward returned at any given time step *t* is the quantity that can be represented as

$$R_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} r(S_{t+\tau}, A_{t+\tau})$$

where $\gamma \in (0,1)$ is the discount factor that multiplies the future expected reward and varies on the range of [0,1]. At any moment, the goal of the DRL agent is to maximize the expected return from each state S_t. The action value indicated by $Q^{\pi}(s, a) = \mathbb{E}[R_t|S_t = s, a]$ is the expected return for taking an action *a* in state *s* by following a policy π. Similarly, the optimal value function indicated by $Q^*(s, a) = max_{\pi} Q^{\pi}(s, a)$ is the maximum action value for action *a* and state *s* that is achievable by any policy. Similarly, the value of any state *s* under policy π is defined by $V^{\pi}(s) = \mathbb{E}[R_t|S_t = s]$ which is simply the expected return for following the policy π from state *s*. The $Q(s, a)$ is often used as a measure of the value of the agent being in that particular state *s* and taking an action *a* to reach that state. The famous Bellman's equation mentioned below is used as a reference to calculate the $Q(s, a)$ for every action in every state that helps an agent to make decisions about its future moves.

$$Q'(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \, maxQ`(s`,a`) - Q(s,a)] \quad (1)$$

where $Q(s, a)$, $Q'(s, a)$, α, $R(s, a)$, γ and $maxQ`(s`, a`)$ represents the current Q value, new Q value calculated, learning rate, the reward for taking that action *a* in state *s*, discount factor($\gamma$) and the maximum expected future reward was given the new state *s`* respectively, with all the possible actions from that state.

In the case of value-based model-free reinforcement learning methods, the action-value function $Q(s, a)$ is often represented by using a function approximation method, such as a neural network. In such a case an approximate action-value function that parameterized with θ represented as $Q(s, a;\theta)$. The updates for the parameters are decided with the help of a suitable RL algorithm. In contrast to the aforementioned value-based methods, policy-based model-free RL methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameter by performing, typically approximate, gradient ascent on $\mathbb{E}[R_t]$.

2.1.2 Markov Property

Formally, the reinforcement learning environment is considered as the mathematic representation of a Markov decision process (MDP) [7]. Major components of MDP consists of the following:

- Set of all possible states that an agent can be while in the environment, represented by *S*.
- Set of all possible actions, *A*, that an agent can take while in a state s € S.
- Transition dynamics function defined as $T(s, a, s') = Pr(St+1 = s'|St = s, A = a)$. Since the actions are considered part of a probability distribution, here *T* represents distribution over the possible resulting state by taking a specific action while in a given state s.
- A reward function, *R*, is associated with a state transition by taking a specific action $R(S_t, a_t, S_{t+1})$.
- A discount factor $\gamma\ [0,1]$, will be used for the calculation of discounted future rewards associated with each state transition. Generally, a low discount factor value will be applied for expected future rewards for state transitions leading to the nearby

states, whereas a high discount factor value will be applied for rewards associated with actions leading to states that are far from the current state [7].

Reinforcement learning models are denoted by the Markov decision process because often such models make the Markov assumption. The core idea behind the Markov assumption is that if one knows the current state one is in, then the history, sequence of actions, and states that took the agent to the current state, does not matter. Going with this key assumption, the core concept underlying each of the RL problems is the Markov decision property—which says that only the current state will have an influence on the next state, and given the current state, the future is independent of the past. In another way, it can be interpreted as any action taken at state S_t can be solely based on the state immediately preceding it, S_{t-1}, but totally independent of all other states $\{ S_0, S_1, ..., S_{t-2} \}$ [8]. In the context of RL, the term policy π is used to define a mapping from a state to a related action that is defined over the probability distribution of actions.

This can be denoted as $\pi(s): S->P_r (A = a|S)$. A policy is considered to be an optimum policy $\pi^*(s)$ for a state s if the specified action taken from that particular state can lead to the maximum expected discounted future reward [7]. In theory, the final objective behind each of the RL agents is to solve the MDP by deducing an optimum policy.

2.1.3 Key Challenges in Reinforcement Learning

Some of the major challenges related to reinforcement learning (RL) can be summarized as follows:

- By heavily relying only on the reward values, an agent needs to follow a brute-force strategy to derive an optimal policy.

- For every action taken while in a particular state, the RL agent needs to deal with the complexities related to the maximum expected discounted future reward for that action. This scenario is denoted as the (temporal) credit assignment problem [9].
- In environments with a 3D nature, the size of the continuous state and action pairs can be quite large.
- Observations of an agent from a complex environment heavily depend solely on its actions, which can contain strong temporal correlations.

2.2 Multi-Task Learning

The traditional learning methodology followed in machine learning is to learn one task at a time. Under this methodology, complex and large problems are broken into small and independent subproblems that are learned separately, then eventually all of this learning is combined towards the overall solution of the problem [10]. There could be occasions in which this approach can be less productive, especially when dealing with complex real-world scenarios (such as autonomous driving systems) that have a source of information with a lot of interdependent tasks. For these kinds of situations, if multiple tasks can learn together and then share their knowledge among themselves, eventually that would make the generalization performance of the overall system increase to a greater extent in comparison to the traditional approach explained above. Multitask learning (MTL) is defined as an inductive transfer mechanism with the key objective to improve generalization performance [11]. The core objective behind multi-tasking is to follow a learning-to-learn methodology to leverage the domain-related information accumulated by training the individual, related tasks in parallel with a shared representation of the system [12]. In this way, the knowledge that is acquired during each task learning can be utilized

and thereby help other tasks be learned better. Eventually, with this approach multitask learning improves the overall generalization performance, which can be applied across many domains including RL and can be used with different learning algorithms within the RL arena. From the perspective of reinforcement learning, multi-task learning is an approach intended to optimize the performance of an agent under the assumption that performance bottleneck problems experienced by the agent are drawn from the same distribution. When it comes to deep reinforcement learning, multi-tasking could be applied from various levels, such as single-agent–multiple tasks and multiple agents–multiple tasks.

2.3 Multi-Task Deep Reinforcement Learning

In recent years, with growth in AI and DL arena, DRL has been merged as the state-of-the-art in many benchmark tasks as well as in real-world problems. Due to this reason, a growing level of attention has been paid to various methods for its optimization as well. The following sections discuss various approaches and techniques developed for multi-task DRL that are presented in related works.

2.3.1 Transfer Learning Oriented Approach

Before the inception of deep learning into the reinforcement learning arena, most of the early research efforts on the development of the multi-task-oriented algorithm within reinforcement learning attempted to use assistance from transfer learning. The core idea behind transfer learning is about transferring knowledge across different but the related source and target tasks to improve the performance of machine learning (ML) algorithms used for learning the target task [13]. Transfer within reinforcement learning predominantly focuses on deriving various methods to transfer knowledge from a set of

source tasks to a target task. This approach has shown good results when the similarity levels within the source and target tasks were similar [14]. If the similarity level between the source and target tasks is quite high, then the transferred knowledge can be quite easily used by the underlying learning algorithm to solve the target task efficiently [15]. This is due to the reason that under such a situation, learning algorithms could achieve optimal performance by leveraging the transferred knowledge rather than relying on more data samples for learning the target task. By leveraging the above methodologies, transfer learning methods have been already applied to single agent-based RL algorithms [16].

There were also research attempts related to extending the same methodologies concerning the multi-agent systems, wherein agents interact with other agents acting in the same environment and then use the knowledge resulting from their actions as well [17]. In general, multi-agent systems are based on a joint policy that the agents learned in the source task (training task), and then use this policy knowledge to formulate the initial policy of the agents in the target task towards the same [18]. Transfer of knowledge is done differently between the source and target tasks with the help of multiple transfer methods, such as for instance transfer, representation transfer, or parameter transfer. In each of these methods, underlying transfer algorithms rely heavily on the prior knowledge learned when solving a set of similar source tasks, and then use it as a reference to bias the learning process on any new task [14].

2.3.2 Learning Shared Representations

Learning the shared representations for value functions is an approach that is quite similar to the transfer learning methodology [19]. This method was developed based on the function approximation capability of neural networks and their application into the

reinforcement learning domain [3]. The major factor behind the success of deep neural networks with reinforcement learning was due to deep learning algorithms' key ability to distill meaningful representations from high-dimensional input states associated with the environment [20]. This key factor scaled up the applicability of RL to more complex environments and scenarios that were previously impossible or demanded a great level of feature engineering [3]. The ability to develop a good abstraction of the environment and the agent's role within that environment are the pivotal factors behind the success of this approach [21]. The core idea behind this approach is based on the assumption that different tasks that an RL agent needs to learn during its life may have a shared structure and in-built redundancy [14]. If these common factors can be abstracted, then they could play a vital role in speeding up the entire learning process. Learning shared representations is a way to achieve this objective through learning robust, transferable abstractions of the environment that generalize over a set of tasks encountered by the agent while in the environment [19].

The value function is one of the key ideas within the RL domain and is being used primarily in conjunction with functional approximators to generalize over large state-action spaces associated with an agent's environment [22]. Value functions are being calculated and used as a key measure to indicate how good a particular state is. Value functions exhibit a compositional structure concerning the state space and goal states [23]. Additionally, earlier researches have shown that value functions can capture and represent knowledge beyond their current goal that can be leveraged or re-used for future learning [22]. By leveraging the state-action value space of common structures shared among different tasks that an RL agent will be handling during its lifetime while in an environment, optimal value functions can be learned. This can be achieved by accommodating the common structure

mentioned above into the popular value iteration and policy-iteration procedures named fitted Q-iteration and approximate policy iteration, respectively.

2.3.3 Progressive Neural Networks

This is an approach quite similar in nature to the transfer learning methodology. This method was developed based on the function approximation capability of neural networks [24]. One of the major challenges associated with the optimization of multi-task learning within the DRL arena was related to leveraging the transfer of learning, and also how to avoid catastrophic forgetting. As a solution to this problem, various researches have been conducted, and one such step forward in this direction is an approach named progressive neural networks. It has the ability to protect itself from catastrophic forgetting and can also leverage prior knowledge with the help of lateral connections to previously learned features. The progressive neural network is a multi-tasking methodology developed by DeepMind using the concept of lateral features transferring that leverages on neural networks [25]. The key characteristic of the model proposed by this methodology is that it possesses the ability to learn new tasks and also maintain previous knowledge learned with the help of progressive neural networks. The idea of having a continuous chain of progressive neural networks is to facilitate the transfer of knowledge across a series of tasks. Conceptually, progressive neural networks have been designed with two major goals. Firstly, have a system with the ability to incorporate prior knowledge during the learning process at each layer of the feature hierarchy. Secondly, develop a system with immunity to a catastrophic forgetting scenario [14].

One of the biggest advantages of this approach is that progressive networks have the ability to retain a group of pre-trained models throughout the entire training cycle [25]. In

addition to this, progressive networks can also learn lateral connections from the pre-trained model to extract useful features for new tasks. This kind of approach with a progressive nature brings richer compositionality, and also allows easy integration of prior knowledge at each layer of the feature hierarchy. This type of continual learning allows the agents to not only learn a series of tasks that are experienced in sequence but simultaneously possess the ability to transfer knowledge from previous tasks to improve convergence speed [26]. Progressive networks integrate these features into the model architecture where catastrophic forgetting is prevented by instantiating a new neural network (a column) for each task that is being solved during an agent's lifetime in the environment. Along with this, knowledge transfer is enabled through lateral connections to the list of features from the previously learned columns [25]. At any timestep, whenever a new task is learned, the model adds a new column of knowledge into its existing framework in the form of a new neural network unit. Further on, this new unit will be used during the learning of successive tasks. Each column (neural network unit) will be trained to solve a particular Markov decision process (MDP) [25]. One of the possible downsides associated with this methodology is that it could be computationally expensive due to its growing size as the learning cycle progresses.

2.3.4 PathNet

PathNet is a multi-task reinforcement learning approach that was developed with the objective of achieving artificial general intelligence (AGI) by combining the aspects of transfer learning, continual learning, and multitask learning [26]. It is based on a neural network algorithm that uses multiple agents that are embedded in the neural network. The objective of each of these agents is to identify which parts of the network to re-use while

learning new tasks [12]. Agents are the pathways (also known as genotypes) within the neural network that determine the subset of parameters that are used during the learning process [27]. These parameters, which are used for the forward propagation of the learning process, often undergo modification during the backpropagation stage of the PathNet algorithm. During learning the learning process, a tournament selection genetic algorithm will be used for the selection of pathways through the neural network. Agents execute actions within the neural network and build the knowledge on how effectively existing parameters in the environment of the neural network can be re-used for new actions (tasks) [28]. Agents often work in parallel with other agents who are learning other tasks and share parameters among them for positive knowledge transfer; otherwise, they update the disjoint parameters that are causing negative knowledge transfer [27].

A PathNet architecture consists of a deep neural network having L layers, with each layer having M modules. Each of these modules will be a neural network. The integrated outputs of the modules from each of the layers will be passed into the active modules in the next layer [14]. In every layer, there will be a maximum number of modules (typically 3 or 4) that are allowed for each of the pathways [27]. The final layer within the neural network of each of the tasks that are being learned is unique and will not be shared with any other task within the environment. One of the advantages of the PathNet is that with this approach a neural network can quite efficiently reuse existing knowledge instead of learning from scratch for each task. This feature could be extremely useful in the context of reinforcement learning, where there are numerous interrelated tasks present in state space. PathNet has exhibited positive results for the knowledge transfer for binary MNIST dataset (Modified National Institute of Standards and Technology), CIFAR-100 dataset

(Canadian Institute for Advanced Research), and SVHN dataset (The Street View House Numbers) supervised learning classification tasks and a set of Atari and Labyrinth reinforcement learning tasks.

2.3.5 Policy Distillation

Policy distillation (PD) and actor-mimic (AM) are the two approaches that leverage the concept of distillation towards achieving multi-task deep reinforcement learning. Distillation is an approach related to minimizing computational costs of ensemble methods [29]. An ensemble is nothing but a set of models whose prediction values are combined by following a weighted averaging or voting method [30]. Ensemble methods have been one of the significant research areas in the past decade, and some of the popular ensemble methods include names such as bagging, boosting, random forests, Bayesian averaging, and stacking [30]. Two of the disadvantages associated with most of the ensembles are that they are often large in terms of memory size needed, and slow due to the time required to execute them at run-time. To cope with these disadvantages, the distillation technique was proposed, which is based on a model compression methodology. The key idea used behind this methodology was to compress the function that is learned by a complex model (often an ensemble) into a much scaled-down, faster model that has comparable performance with the original ensemble [30]. Later on, the same methodology was mapped into the neural networks' domain [31].

By following the concept of model compression that was explained above, policy distillation can be viewed as a technique used to extract the policy of a reinforcement learning agent [32]. Further on this policy will be used to train a new network that performs at the expert level with a smaller size and with higher efficiency [33]. Furthermore, the

same methodology can be extended to consolidate multiple task-specific policies into a single policy for the RL agent. Early researches of policy distillation were done with the reinforcement learning algorithm named DQN (deep Q-network). The policy distillation technique was successfully used for transferring one or more active policies from deep Q-networks to an untrained network [14]. DQN is one of the popular state-of-the-art model-free approaches used for reinforcement learning by using deep neural networks, which operates within an environment with discrete action choices. This algorithm was shown to surpass human-level performance on a group of diverse Atari 2600 games [3]. Distillation can be applied both at a single task level (single game policy distillation) as well as a multi-task level as a knowledge transfer method from a teacher model *T* to a student model *S*. Under the single task policy distillation, data generation will be done by the teacher network (a trained DQN agent), and further supervised training will be carried out by the student network. In order to achieve multi-task policy distillation, *n* different DQN-based single-game experts (agents) are trained separately [30]. After this, these agents individually generate the inputs and targets and store these data in different memory buffers. Further on, the distillation agent learns from these n data stores sequentially.

2.3.6 Actor-Mimic

One of the key aspects of an intelligent agent is its capability to act in multiple environments and transfer the knowledge accumulated from past experiences to new situations. Actor-mimic is such an approach that mainly concentrates on multitask and transfer learning aspects. These capabilities enable an intelligent agent (RL agent) to learn how to act with multiple tasks simultaneously and then generalize this accumulated knowledge to new domains [34]. In general, actor-mimic can be viewed as a method for

training a single deep policy network by using a group of related source tasks. A model that was trained with this method was found to reach expert-level performance on many games. More importantly, with a significant level of similarity between the source and target tasks, features that are learned during the training of the source tasks can be used well for generalization while training the target tasks [35].

The actor-mimic approach leverages both deep reinforcement learning and model compression techniques to train a single policy network. The objective of such training is to make the network learn how to act in a set of distinct tasks under the guidance of several expert teachers [14]. Further on, representations learned by this deep policy network can be used for generalizing to new tasks with no prior expert guidance. This approach was predominantly tested within the arcade learning environment (ALE) [36]. Often, actor-mimic is treated as part of the larger imitation learning class of methods that are based on the idea of using expert guidance to teach an agent how to act within an environment. Under the imitation learning methodology, a policy will be trained to directly mimic an expert's behavior during sampling the actions from the mimic agent [34].

2.3.7 Asynchronous Advantage Actor-Critic

A3C (asynchronous advantage actor-critic) is an algorithm that was introduced by DeepMind, which proposed a parallel training approach. As per this methodology, there will be multiple agents (also known as workers) that are executing in parallel on multiple instances of the same environment [4]. These multiple workers running in parallel environments update a global value function in an asynchronous fashion. During the training, at any particular time-step *t*, all these parallel agents will be experiencing a variety of different states, which almost makes the learning of all agents unique. As a result of this

uniqueness factor, A3C provides agents with an effective as well as an efficient exploration of the entire state space within the environment [37]. Originally, A3C was an extension of the actor-critic method, wherein there will be two independent neural network components named actor and critic, each with its loss functions [14]. An actor can be considered as a function approximator that guides on how to act, as it is being judged by RL methods, such as Q-learning or in REINFORCE. In both of these methods, a neural network will be computing either a function that leads to a policy or directly calculating the policy itself [38]. The role of the critic is more like evaluating the effectiveness of the policy made by the actor and giving feedback for further enhancement of the policy [4].

The subsections that are covered under Section 2.3 have provided details on various approaches and techniques that are developed for facilitating multi-task learning within the reinforcement learning domain. To this end, this chapter has provided a platform to understand the existing approaches attempted so far. Next, in section 2.4 we would discuss the related state-of-the-art research efforts conducted within the area of multi-task deep reinforcement learning by the research organizations such as Google DeepMind and OpenAI.

2.4 Related Work

This section provides the details on the related work done on the multi-task DRL front. Before the inception of deep reinforcement learning, most of the multi-task-oriented algorithms relied on transfer learning to realize proper control over different tasks. Besides, some research efforts were carried out to investigate the joint training of multiple value functions or policy functions over a set of tasks [39] [40]. However, the functionalities of all of these algorithms were limited by handcrafted features. Even though a huge amount

of work has been done to improve DRL algorithms over single tasks, relatively there is much less amount of work done for multi-task scenarios. Some of those research attempts either focused on the exploration and generative models or explored learning universal abstractions of state-action pairs or feature successors, which are quite similar in nature to transfer learning methodology [41]. DiGrad (Differential Policy Gradient) is an approach developed for simultaneous training of multiple tasks sharing a set of common actions in continuous action spaces. The proposed framework is based on differential policy gradients and can accommodate multi-task learning in a single actor-critic network. This framework was designed predominantly for efficient multi-task learning in complex robotic systems and tested on 8 link planar manipulators and 27 degrees of freedom (DoF) Humanoid for learning multi-goal reachability tasks for 3 and 2 end effectors respectively [42]. Another research work related to the multi-task learning done was based on the model-based approach to deep reinforcement learning which we use to solve different tasks simultaneously. This model was developed with a recurrent neural network inspired by residual networks that decouple memory from computation allowing to model complex environments that do not require lots of memory [5]. Another relevant work at the multi-task front done was mainly attempting to address the partial observability issue of RL with help of the deep decentralized multi-task multi-Agent reinforcement learning method. It was based on a decentralized single-task learning approach that is robust to concurrent interactions of teammates and presented an approach for distilling single-task policies into a unified policy that performs well across multiple related tasks, without explicit provision of task identity [43]. Diffusion-based Distributed Actor-Critic (Diff-DAC) is a deep neural network-oriented distributed actor-critic algorithm designed to single-task and to average

multitask reinforcement learning (MRL). In this method, each individual agent is having access to data from its local task only, and during the learning, process agents share their value-policy parameters with neighbors to converge to a common policy but without having a central node [44]. For the remainder of this section, we will focus on comparing and contrasting the three state-of-the-art approaches namely DISTRAL, IMPALA, and PopArt.

2.4.1 DISTRAL

DISTRAL (DIStil and TRAnsfer Learning) is a new approach that was developed for multi-task training by designing a framework with the objective of simultaneous reinforcement learning of multiple tasks [6]. The major design focus was on building a general framework for distilling the centroid policy and then transferring common behaviors of individual workers in multi-task reinforcement learning. Instead of the parameter sharing among the various workers within the environment, the key idea behind the design of DISTRAL was to share a distilled policy that can capture common behavior across tasks. After deducing the distilled policy, further on, it will be used to guide task-specific policies through regularization by using a Kullback-Leibler (KL) divergence [34]. In this way, firstly knowledge gained in one task is distilled into the shared policy, and then finally transferred to other tasks. With this approach, each worker will be individually trained to solve its task by staying as much close as possible with the shared policy. This shared policy will be trained by using the distillation which acts as the centroid of all task policies [45]. This method was proven to be quite efficient for the transfer of knowledge on RL problems that involve complex 3D environments.

DISTRAL approach has proven to be outperforming the traditional way of using shared neural network parameters for multitasking or transfer reinforcement learning by a huge margin. This was predominantly due to the two major factors listed below. Firstly, distillation plays a vital role in the optimization procedure when using KL divergences as a means to regularize the output of task models towards a distilled model deduced from the individual task policies. Secondly, usage of the distilled model itself as a method to regularize for training the individual task models within the environment. More importantly, using the distilled model as a method to regularize, brings the aspect of regularizing the networks (of individual workers) in a more meaningful space such as task policies than at the parameters' level [27].

2.4.2 IMPALA

IMPALA (Importance Weighted Actor-Learner Architecture) approach proposed by Google DeepMind is a distributed agent architecture developed by adopting a single reinforcement learning agent having a single set of parameters. One of the key aspects of the IMPALA approach is its ability to effectively use the resources in a single-machine training environment, while it can also be scaled to multiple machines without sacrificing data efficiency or resource utilization. By leveraging on a novel off-policy correction method named V-trace, IMPALA can achieve quite stable learning at high throughput by combining decoupled acting and learning [46]. Typically, the architecture of a deep reinforcement learning model is based on a single learner(critic) that is combined with multiple actors. In this ecosystem, every individual actor generates its learning cycle parameters (also known as trajectories), and then sends that knowledge to the learner (critic) through a queue. The learner collects the same kind of trajectories from all the other

actors in the environment and prepares a central policy. Before the next learning cycle (trajectory), every actor retrieves the updated policy parameters from the learner(critic) [46]. This approach is quite close to the reinforcement learning algorithm named A3C, and the architecture of the IMPALA has been heavily inspired by this RL algorithm. IMPALA's model follows a topology of multiple actors and learners can collaborate to build knowledge.

IMPALA follows an actor-critic setup to learn a Policy π and, a baseline function named $V\pi$. Major components of the IMPALA system consist of a set of actors who continuously generating trajectories of experience, and additionally, there could be one or more learners that use these generated experiences sent from actors to learn π, which is an off-policy. At the start of each trajectory, an actor will update its local policy μ to the latest learner policy π. Further on the actor will run that policy for n steps in its environment. At the end of these n steps, the actor sends another trajectory of states, actions, and rewards together with related policy distributions to the learner. In this manner, the learner will be continuously updating its policy π each time when trajectory information is collected from the actors within the environment [46]. In this manner, IMPALA architecture collects experiences from different learners which are passed to a central learner. Further on this central learner calculate the gradients, and generates a model with independent actors and learners. One of the key aspects of this IMPALA architecture is that actors need not be present on the same machine, but can be distributed across many machines.

2.4.3 PopArt

As an attempt to enhance the issues associated IMPALA model, DeepMind proposed a new method named PopArt to improve reinforcement learning in multi-task environments.

The core objective of PopArt is to minimize the distraction dilemma problem associated with the IMPALA model and, thereby stabilize learning to facilitate the adoption of multi-task reinforcement learning techniques [47]. Distraction Dilemma refers to the probability of learning algorithms getting distracted by a few tasks from the set of multiple tasks to solve. It often leads to the challenge related to establishing a balance between the needs of multiple tasks within the same environment competing for the limited resources of a single learning system. The PopArt model is designed based on the original IMPALA architecture model with the combination of multiple convolutional neural network layers with other techniques such as word embeddings with the help of a recurrent neural network of type long-short term memory (LSTM) [47].

PopArt methodology works by adapting the contributions from each of the individual tasks to the agent's updates. This way PopArt ensures that all agents will have their role and a proportional impact on the overall learning dynamics. The key aspect of the PopArt relies on modifying the weights of the neural network based on the output of all tasks within the environment. In the initial stage, PopArt estimates the mean as well as the spread of the ultimate targets such as the score of a game across all tasks under consideration. Further on PopArt use these estimate values to normalize the targets before updating the network’s weights. This approach makes the learning process more stable and robust. With the experiments conducted with Atari 2600 games, PopArt has demonstrated improvements over other multi-task reinforcement learning architectures [47].

Table 2.1. Comparison of State-of-the-art Solutions.

Details of Feature	**Name of Solution**		
	DISTRAL	IMPALA	PopArt
Distraction Dilemma problem	No	Yes	No
Master Policy-based Framework	Yes	Yes	Yes
Operation Methodology	Policy Distillation	Actor-critic model-based distributed system	Extension of IMPALA with support for RNN such as LSTM
Multi-task Learning Support	Yes	Yes	Yes

Optimization of RL agent's performance is an active area of research and there has been a growing amount of literature detailing the various approaches attempted to increase the performance levels of RL-based intelligent agents. The related work is here focused on three state-of-the-art multi-task learning frameworks from Google DeepMind, but there are research papers belonging to different domains that have utilized the ideas from other related research efforts. A deep reinforcement learning optimization framework (DRLOF) is a method to determine the optimal operating conditions for a commercial circulating fluidized bed (CFB) power plant that strikes a good balance between performance and environmental issues. The DRLOF included the CFB as an environment created from over 1.5 years of plant data with a 1 min sampling time which interacted with an advantage

actor-critic (A2C) agent of two architectures named ‘separate-A2CN’ and ‘shared-A2CN’. The framework was optimized by maximizing electric power generation within the constraints of the plant’s capacity and environmental emission standards, taking into consideration the cost of operations [48]. Deep Reinforcement Learning (DRL) has recently spread into a range of domains within physics and engineering, wherein DRL has been used to direct shape optimization [49]. An artificial neural network trained through DRL is able to generate optimal shapes on its own, without any prior knowledge, and in a constrained time. Trading strategies are well depicted as an online decision-making problem involving imperfect information and aiming to maximize the return while restraining the risk. There have been researching efforts focusing on designing a rule-based policy approach to train a deep reinforcement learning agent for automated financial trading. During this multiplex process, the agents which are trained on 504 risky datasets, use the fundamental concepts of proximal policy optimization to improve their own decision-making by adjusting their action choice against the uncertainty of states [50]. Another research work done related to HVAC proposed a data-driven DRL-based method to minimize HVAC users’ energy consumption costs while maintaining users’ comfort. The applied DRL method's efficiency is enhanced by conducting multi-task learning that can achieve an economic control strategy for a multi-zone residential HVAC system in both cooling and heating scenarios [51]. There were efforts that attempted to achieve optimization by attempting to combine Q-learning achievements with DISTRAL's multi-task learning capabilities into hybrid architecture called Rainbow. This architecture proposed a method to achieve higher performance in multi-task DRL scenarios by adopting a rainbow agent by leveraging the DISTRAL framework [52]. Another research effort

named Auto-Agent-Distiller (A2D) framework, adopted a neural architecture search (NAS) method applied to DRL to automatically search for the optimal DRL agents for various tasks that optimize both the test scores and efficiency [53].

2.4.4 Summary

The topics mentioned as the subsections under section 2.4 have given information on the state-of-the-art research efforts conducted within the multi-task deep reinforcement learning by presenting state-of-the-art models such as DISTRAL, IMPALA, and PopArt. In addition, this section has also listed out various research efforts from literature, that are done towards the optimization of RL intelligent agents. In the next section, chapter 3, we discuss the design methodology and other related aspects of the proposed hybrid multi-task learning model by leveraging on the A3C model.

Chapter 3

Hybrid Multi-Task Learning Model

This chapter describes the design aspects of the proposed multi-task learning model and its architecture. To this end, the chapter first presents the overview of the hybrid A3C model and further explains the actor-critic methodology. Along with this, the role of both actor and critic modules in the overall learning process is also explained. Finally, details of the A3C are provided to explain how the parallel multi-task learning capabilities of actor-critic methodology could be adopted into the hybrid multi-tsk learning model.

3.1 Hybrid A3C Model

The major motivation behind the proposed hybrid multi-task approach is to address and mitigate some of the key challenges associated with DRL multi-tasking, which are not fully covered by the state of the art. Our proposed approach would be attempting to address the optimization bottlenecks posed by challenges such as partial observability, effective exploration, and also the amount of training time needed to achieve acceptable levels of performance [54].

The proposed approach named the hybrid A3C model is an attempt to address most of these aspects, by extending the basic actor-critic model to two different environments with a high level of semantic similarity. Within the context of the proposed hybrid multi-task learning model, the notion of semantic similarity relies on two key aspects. The foremost aspect is related to the high level of similarity in terms of the actions that DRL agents in both environments are going to take at any state *s*. For, instance, DRL agents' actions in both environments could be from a common 4 member-tuple {up, down, right,

left}, which will decide the state transition probability. The impact of the aforementioned factor will become more prevalent when the nature of the agents' rewarding tasks in both environments are also the same. For instance, if we consider two games namely Breakout-v0 and Pong-v0 from the Atari-57 family, the aim of the DRL agents operating in both environments is to balance a paddle to hit a ball by choosing one of the actions from the above mentioned 4-member tuple. Another example could be a game pair such as DemonAttack-v0 and SpaceInvaders-v0, where the aim of the agent is to balance a ship and shoot the enemies. The decision to choose the A3C algorithm for building the proposed hybrid multi-task learning model was after the careful examination of few factors that were lagging within the related works analyzed. To start with, the optimization of the DRL agent's performance is more challenging whenever state-action space is massive, which is the case with a model-free environment. As the DRL will be heavily relying on the high representational learning power of DL, it is highly important to have the neural network with many balanced weights that could lead to better agent policy. To achieve this objective, gradient-based knowledge sharing is the optimal choice as it would help to balance the weight matrix of underlying neural networks. Secondly, an on-policy agent setting is the most preferred way for the DRL agent to derive an optimal policy in less amount time in comparison to the off-policy agent setting. Lastly, coupling the aspect of transfer learning with multi-task learning would boost the learning speed of DRL agents when there is a semantic similarity between the tasks executed within the multiple environments. Having these details, the A3C algorithm was found to be the right choice to achieve these objectives, especially with its actor-critic-based design. An algorithm such as DQN (Deep Queue Network) was not able to meet the above-mentioned aspects as it

was having an off-policy-based agent setting. DRL agent functioning with an off-policy-based algorithm such as DQN could be selecting the actions randomly, which could further lead to more training time to optimize the agent's performance. The key aspect the of A3C algorithm is its ability to learn multiple instantiations of a single target task simultaneously, and also its ability to improve the model's performance by transferring the knowledge between multiple instantiations [4]. From the perspective of a model-free environment with massive state space, A3C's multi-threaded-based working model would increase the momentum of DRL agents' performance optimization. The proposed hybrid A3C approach will be leveraging this key aspect and will attempt to achieve this objective across, two different by semantically similar environments with related tasks. The hybrid approach will be heavily relying on the applicability of the multi-threaded capability of the A3C algorithm across semantically related tasks running in two different environments. In addition to this, as A3C is based on the actor-critic methodology, this allows the agent to directly derive the policy to decide the action to take at each state. When it comes to algorithms such as DQN, which is based on the notion of the Q-value function, the selection of the action will be based on Q-value. In relation to this, actor-critic-based A3C gives an added advantage over Q-value function-based algorithms such as DQN. A3C-based learning could be more complex with a very high number of workers running with the possibility of negative knowledge or gradient transfer. The proposed approach could be treated as a model running two threads of the A3C algorithm, wherein each thread will be managing the multiple instantiations of the tasks running in each environment. Each of these individual threads would consider itself as a subtask such as A and B, with each of them sharing its individual learning with the learner in an asynchronous manner. Further

on, the learner (global network) will be converging the knowledge from both of these threads and deducing a new policy, that will be applied back on the threads. The key aspect of the hybrid approach is only to enhance the performance of the RL agent through a joint-learning through multi-task learning approach by using deep reinforcement learning. Fig. 3.1 shows the high-level architecture model of the proposed hybrid multi-task approach.

The hybrid A3C model deploys multi-threaded asynchronous variants of the advantage actor-critic algorithm. The major objective behind designing this model is to find a methodology that can train deep neural network policies reliably and without large resource requirements. During the development of the hybrid A3C model, initially, we conducted its validation on a desktop-based environment which is having a dual-core CPU on a single machine. Under this environment, we have conducted basic level testing with a pair of actor-learner worker threads. With this, one (actor-learner) worker thread was assigned to run the task from each game's environment. Over the course of the execution, this model asynchronously attempts to derive and optimize the global policy based on the observations that multiple actors-learners running in parallel are likely to be exploring different parts of the environment. At an individual actor-learner module level, it is possible to have different exploration policies in each module to maximize this diversity. In this way, having different exploration policies in different threads of the actor-learner module, the overall changes being made to the global network parameters by these different actor-learners applying asynchronous updates in parallel are likely to be less correlated. This model is designed to run on a single machine with a standard multi-core CPU and applied to a variety of Atari 2600 domain games for testing.

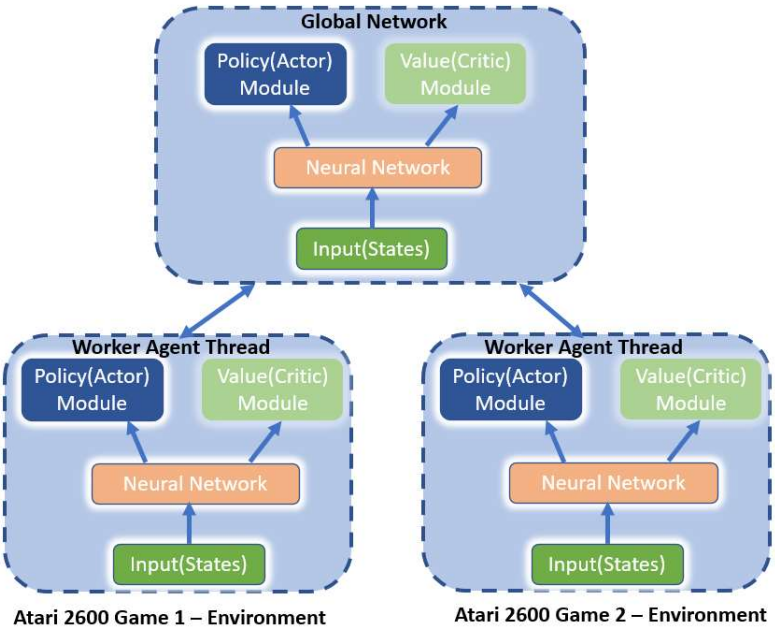

Figure 3.1: The architecture of the hybrid multi-task learning model.

The semantic similarity aspect of the related tasks running two different gaming environments is the most vital factor to achieve the above-mentioned objectives, which otherwise give challenges in terms of negative knowledge transfer. Negative Transfer is considered to be one of the key challenges while dealing with the multi-tasking aspect within the reinforcement learning domain. The main idea of knowledge transfer learning in a multi-task context is that transferring knowledge accumulated from learning from a set of source samples under one agent may improve the performance of another task agent while learning on the target task [34]. However, this knowledge transfer could impact the overall learning progress and performance of the agent in either way, positively or negatively. If there is a considerable difference between the source tasks and target tasks, then the transferred knowledge could create a negative impact.

Having multiple environments with a high level of semantic similarity would indirectly improve the partial observability by exchanging the learning across the agent's operating environment [55]. Similarly, having multiple actor-critic models operating simultaneously across two semantically similar environments would mitigate RL agent's issues associated with effective exploration, and the training time required to reach an optimized performance level [56].

3.2 Actor-Critic Methodology

Unlike some simpler techniques which are based on either value-iteration (Q-learning) methods or policy-gradient (PG) methods, the actor-critic (AC) methodology combines the best parts of both the methods, which are the algorithms that predict both the value function V(s) as well as the optimal policy function π(s) [57]. In other words, actor-critic methods consist of two models, namely an actor module and a critic module. Thereby AC attempt to combine the aspects of both policy gradient and value gradient into a single model. Fig. 3.2 shows the diagram of actor-critic methodology.

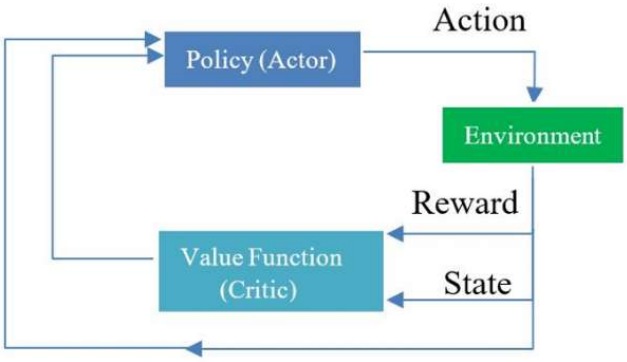

Figure 3.2: Actor-Critic model.

The actor acts as a policy network, that decides for a given state which action *a* to be taken at each given time step *t*. The critic consists of a value network *Vπ(s, a)* that tells how

promising action is under the current state *s*. Having said that, in its role critic outputs an evaluation value *V(s, a)* for the actor, which indirectly helps the actor to adjust its policy for better results. At the same time, both actor and critic networks update themselves according to the knowledge gathered by their respective neural networks from the environment. This internally helps the agent to converge its policy to the optimal policy π_θ^*. In summary, the critic module updates the value function parameters *w*, and depending on the algorithm it could be either action-value $Q_w(a|s)$ or state-value $V_w(s)$ whereas the actor module updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.Fig.3.3 shows the single actor-critic worker agent flowchart [38]. The learning agent uses the value from the value function calculated by the critic module to update the optimal policy function of the actor module. Note that here the policy function means the probabilistic distribution of the action space. To be exact, the learning agent determines the conditional probability $P(a|s;\theta)$ which otherwise means parametrized probability that the agent chooses the action *a* when in state *s*.

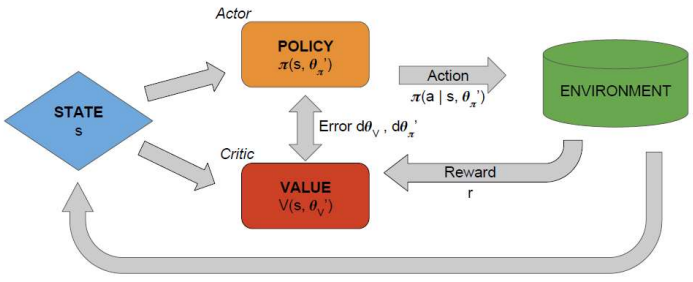

Figure 3.3: A single thread of actor-critic worker execution.

The policy is often modeled as a function $\pi_\theta(a|s)$ that is parameterized to θ. The value of the DRL agent's reward function depends on this policy, and the algorithms are used to

optimize θ. The reward function is defined as below, wherein $d^{\pi}(s)$ notation refers to the stationary distribution of Markov chain for π_θ (for the on-policy state distribution under π).

$$J(\theta) = \sum_{s\varepsilon S} d^{\pi}(s) V^{\pi}(s) \quad (1)$$

$$= \sum_{s\varepsilon S} d^{\pi}(s) \sum_{a\varepsilon A} \pi_\theta (a|s) Q^{\pi}(s,a) \quad (2)$$

Within the AC model, the critic is in charge of updating the value function parameters w, and based on the DRL algorithm it could be either an action-value function $Q_w(a|s)$ or state value function $V_w(s)$. Based on the details of the value function shared by the critic, the actor updates the policy parameter θ for the $\pi_\theta (a|s)$. The execution of an actor-critic algorithm can be explained by the below steps [58].

1. Initialize *s,θ,w* at random, and sample $a \sim \pi_\theta (a|s)$
2. For $t = 1 \ldots T$:
    a. Sample reward $Rt \sim R(s,a)$ and next state $s` \sim P(s`|s,a)$;
    b. Then sample next action $a` \sim \pi_\theta (a`|s`)$
    c. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta \; Q_w (s,a) \nabla_\theta ln \pi_\theta (a|s)$;
    d. Compute the correction (TD error) at time t for action-value:
        i. $\delta_t = r_t + \gamma Q_w(s`|a`) - Q_w(s,a)$
        ii. Use it to update the parameters of the action-value function as given $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
    e. Update $a \leftarrow a`$ and $s \leftarrow s`$

Both the learning rates α_θ and α_w, are predefined for policy and value function parameter updates respectively.

3.3 Actor

An actor is a module that controls how a policy-based DRL agent behaves within an environment. The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy π^*.Policy-based algorithms such as Policy Gradients (PG) and REINFORCE try to find the optimal policy directly without the Q -value as the intermediate step [59]. Often an actor could be a function approximator such as a neural network with its objective as to identify the best action while a DRL agent is in a state S_t at time step t. The neural network could be either fully connected or a CNN.

3.4 Critic

The critic, on the other hand, evaluates the action by computing the value function (value-based). The role of the critic is to evaluate how good an action is taken by the agent with the help of a value-based approach. As in the case of the actor, the critic also could be a function approximator such as a neural network. The result is that the overall architecture will learn to play the game more efficiently than the two methods separately.

3.5 Asynchronous Advantage Actor-Critic(A3C)

A3C is a state-of-the-art DRL algorithm developed based on the AC methodology. This algorithm is designed to function both in discrete and continuous action space environments and can be treated as the multi-thread version of the original AC algorithm. A3C makes the AC algorithm converge faster by running multiple agent threads [33]. Each of these threads consists of an independent actor-critic pair that interacts with the environment simultaneously. The agents, which are also known as workers, are trained in parallel and update periodically a global network, which holds shared parameters. The

updates are not happening simultaneously and that's where the asynchronous comes from. The unique exploration experience offered by each of the global actor-critic networks. With such multiple threads sharing the experience with a global network in an asynchronous fashion, A3C eliminates the bias of continuous experience trajectory by feeding only a small batch of experience tuple *(s, a,r,s`)* at any time. After each update, the agents reset their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again. With this approach, the information flows not only from the agents to the global network but also between agents as each agent resets its weights by the global network, which has the information of all the other agents. Fig. 3.4 depicts the ecosystem of a single actor-critic worker.

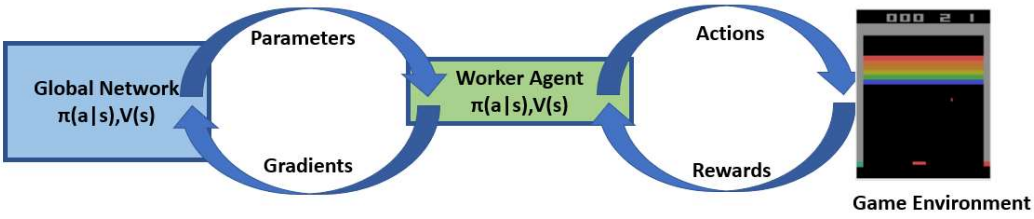

Figure 3.4: The ecosystem of single A3C worker thread with Atari 2600.

A3C uses a deep neural network to model both a policy network $\pi(a_t|s_t;\theta)$ and a value network $V(s_t;\theta)$. For a given state S_t, the policy network (which is the "actor") predicts the optimal action to take at S_t while the value network (which is the "critic") approximates the future reward from taking the optimal action at S_t. By theory, these two networks are separate, but in practice, we use the same convolutional layers for both the

policy and value networks with separate output layers at the end. The Asynchronous nature of A3C means that multiple actor-critic threads are running at the same time, each with its environment. Each thread steps through its environment with its own local CNN, periodically updating a globally shared CNN wherein all networks have an identical architecture. For each thread, at every t_{max} local steps or when a terminal state is reached, that thread syncs its local parameters with the global parameters, computes gradients, and applies them upstream to the global network [60].

A3C follows online learning by adopting a policy gradient method, directly from the states as they are processed by each worker agent thread. The policy is developed naturally as each thread runs within its stochastic Atari 2600 based gaming environment and updates to the global parameters. This methodology suggests that A3C does not overfit to any particular state trajectory of a specific worker thread. Fig. 3.5 shows the worker agent thread's architecture with CNN modules.

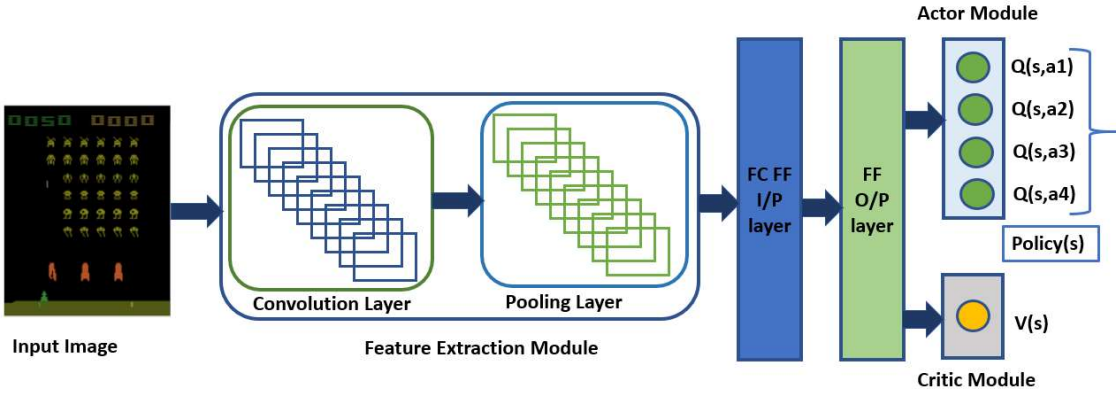

Figure 3.5: The architecture of the worker agent thread in A3C.

The notion of Advantage *A* is used to measure the difference between the expected reward and estimated reward. By using the value of advantage instead, the agent also learns how much better the rewards were than its expectation. This gives a new-found insight to the agent into the environment and thus the learning process is better. The advantage metric is given by the following expression

Advantage: $A = Q(s, a) - V(s)$ (3)

where Q refers to the Q value calculated by the critic module based on the actual reward and TD error following an actor's policy-based chosen action. The Advantage function named $A(S_t a_t; \theta, \theta_v)$ is calculated that needs to be discounted future rewards accumulated to t_{max} or at the terminal state.

$$A(S_t a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \quad (4)$$

Gradients associated with both policy and value networks are denoted by the following equations (5) and (6) respectively, which are calculated by summing over all the states in the past t_{max} local iterations of each worker agent thread's execution [60].

$$\nabla_{\theta`} log\pi(a_t|s_t; \theta) A(s_t, a_t; \theta, \theta_v) \quad (5)$$

$$d\theta = d\theta + \partial(R - V(s_t; \theta`_V))^2 \ / \partial\theta` \quad (6)$$

The pseudocode of the A3C algorithm for each worker agent thread within the hybrid multi-task model is given by the algorithm mentioned below [4].As the design of the hybrid multi-task learning model is having multiple worker agents running within the environment, multiple instances of the same algorithm will be utilized. The framework of the algorithm used within the hybrid multi-task learning model is having the format of

actor-critic methodology based asynchronous learning technique created by Google DeepMind. Each instance of the A3C algorithm for each individual worker threads maintains a policy $\pi(a_t|s_t;\theta`)$ and an estimate of the value function $V(s_i;\theta_v`)$. The values of both policy and value functions are updated after having t_{max} number of actions by the agent or when the terminal state is reached. There will be a CNN with a Softmax output for creating the policy $\pi(a_t|s_t;\theta`)$.

A3C Algorithm – pseudocode for each actor-leaner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ^1 and $\theta_v`$
Initialize thread step counter $t \leftarrow 1$
repeat
Reset gradients $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
Synchronize thread-specific parameters $\theta` = \theta$ and $\theta_v` = \theta_v$
$t_{start} = t$
Get state s_t
repeat
perform a_t according to the policy $\pi(a_t|s_t;\theta`)$
Receive reward r_t and new state s_t
$t \leftarrow t + 1$
$T \leftarrow T + 1$
until terminal s_t **or** $t - t_{start} == t_{max}$
$R = \{0$ terminal s_t
$R = \{V(s_t, \theta_v`)$ for terminal s_t //Bootstrap from last state
for $i \in \{t-1 \ldots . . t_{start}\}$ **do**
$R \leftarrow r_i + \gamma R$
Accumulate gradients wrt $\theta`: d\theta \leftarrow d\theta + \nabla_{\theta`} log\pi(a_i|s_i;\theta`)(R - V(s_i;\theta_v`))$
Accumulate gradients wrt $\theta_v`: d\theta_v + \partial(R - V(s_i;\theta_v`))^2 \ / \ \partial\theta_v$
end for
Perform an asynchronous update of θ using $d\theta$ and of θ using $d\theta_v$
until $T > T_{max}$

3.6 Summary

This chapter has presented the details of the design methodology that is followed towards the development of the hybrid multi-task model, and various aspects of the A3C algorithm that will be leveraged. To this end, it provides the applicability of the basic A3C approach towards the optimization of the DRL agent. In the next section, chapter 4, will be presented with the details on the implementation of a hybrid multi-task learning model with various machine learning libraries and related tools.

Chapter 4

Implementation

This chapter describes the implementation of the proposed hybrid multi-task model which is based on the A3C algorithm. The presented hybrid multi-task model is implemented using multiple tools, and various libraries related to data science, machine learning, and deep learning. This section of the thesis details all those components along with their role in the presented framework.

4.1 Prototype Overview

Throughout the implementation, the prototype was tested with various games under the Atari 2600 environment provided within the OpenAI Gym [61]. The Gym library is a toolkit made by OpenAI for developing and comparing RL algorithms. The first stage of the hybrid multi-task model was constructed by adopting the A3C algorithm for the gaming environment Breakout-v0. The high-level architecture of the model is based on the actor-critic methodology. In our context, the actor is a neural network that parameterizes the policy (π (a | s) and critic is another neural network that parameterizes the value function V(s). The policy network outputs the policy (π), based on which the actor chooses an action within the environment, and the value network outputs the value function V(s). Each of these networks has its respective weights which are often represented by notations such as θp and θv.

$$\pi\ (a|s,\ \theta p) = \textit{Neural Network}\ (\text{input: s, } \textit{weights}\text{: } \theta p) \tag{1}$$

$$V\ (s,\ \theta v) = \textit{Neural Network}\ (\text{inputs, } \textit{weights}\text{: } \theta v) \tag{2}$$

A more graphic intense Atari 2600 gaming environment will be used to execute the hybrid multi-task learning model as it offers a relatively complex environment having an infinite number of state-action spaces to deal with. In order to accommodate and handle this environment, the CNN-based deep neural network model was used for the validation.

4.2 Multi-Task Worker Agent Model

At the root level, this environment will employ a pair of CNN models to implement both actor and critic modules for a single worker. There will be multiple instances of the CNN class objects to implement the multiple worker threads used within the multi-task model. Similarly, the global network was also deployed as a pair of CNN to support the implementation of actor-critic modules at the global network level.

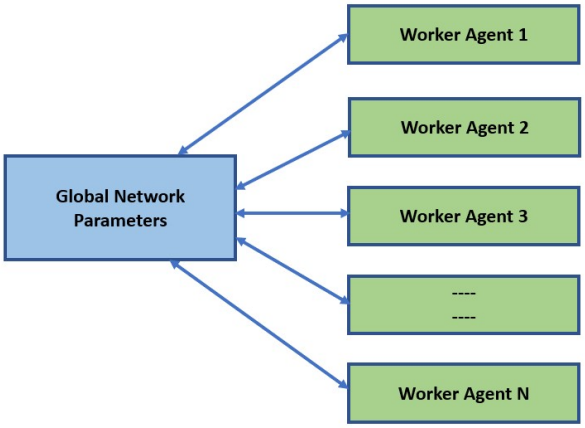

Figure 4.1: A3C multi-task worker agent model.

Fig. 4.1 shows the high-level architecture view of the multi-task model having N worker threads of execution coordinated and managed by a global network. Each of these individual blocks is made up of a pair of CNN networks, each for the actor(policy) and critic (value function) modules. In other words, A3C utilizes N worker agents attacking the

same game environment while being initialized differently. This indirectly points out that each of these agents starts at a different point in their environment so they will go through the same environment in different ways to solve the same problem.

4.3 Training Workflow of Worker Agent

Within the A3C-based multi-task worker agent environment, each of the individual worker agents is managed by the global network directly. Under this scheme, initially, each of the workers is reset with parameter values shared by the global network, later on, the worker interacts with its own individual copy of the environment. Even though each of the worker agents is operating within the same game environment, they are being initialized differently. This gives an opportunity for each of these agents to start at a different point in their environment. Fig. 4.2 shows the training workflow of each agent with the global network.

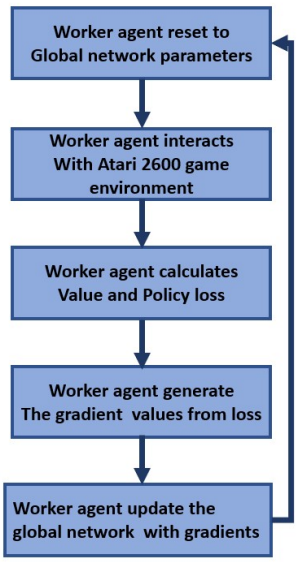

Figure 4.2: Training workflow of worker agent thread.

During the course of its operation, each worker agent plays a fixed number of game episodes and calculates the value and respective policy loss. As these modules, both actor and critic are implemented using the neural network, gradient values are calculated from the losses incurred during its operation. These gradient values will be shared with the global network after the work agent finishes a fixed number of game episodes.

The algorithm behind the operation of the A3C multi-task worker agents' model is mentioned below.

Algorithm of A3C-based multi-task model worker agent.

---

while not done:

$$\boldsymbol{a = sample\ an\ action\ a \sim \pi_\theta(a|s)}$$

$$\boldsymbol{s`, r, done = Perform\ action\ a - env.step(a)}$$

$$\boldsymbol{G = r + \gamma V(s`)}$$

$$\boldsymbol{L_p = -(G - V(s))\log\left(\pi(a|s, \theta_p)\right)}$$

$$\boldsymbol{L_v = (G - V(s))\text{^}2}$$

$$\boldsymbol{\theta_p = \theta_p - \alpha * d_{L_p} / d_{\theta_p}}$$

$$\boldsymbol{\theta_v = \theta_v - \alpha * d_{L_v} / d_{\theta_v}}$$

During the operation, each of the worker agents loops through each step of the game, and samples the action, and updates the weights of both the neural networks- actor and critic. The algorithm runs until a preset number of episodes of the game are played, wherein initially an action *a* is sampled from the actor (policy network). Further on, upon completion of that action respective reward (*r*) and new state (*s'*) are calculated. Based on the new state reached, the total discounted future return (*G*) is calculated by applying the discount factor (gamma). Based on this each of the individual neural networks calculates

its policy loss (Lp), and value loss (Vp) [62]. Further on, the neural network uses gradient descent to update the respective network weights (Θp – policy network weight and Θv – value network weight) to minimize the loss.

4.4 Architecture of Worker Agent

At the root level, this environment will employ a pair of convolutional neural network (CNN) models to implement both actor and critic modules for a single worker. There will be multiple instances of the CNN class objects to implement the multiple-worker threads used within the multi-task model. Similarly, the global network is also deployed as a pair CNN to support the implementation of actor-critic modules at the global network level. These neural network models act as a function approximator by processing each screenshot of the game as its input. We have used RMSprop optimizer with this implementation [63]. The role of the RMSprop optimizer lies in optimizing the neural work weights for both policy and value networks of a hybrid multi-task model. Neural network's weight optimization happens during the backpropagation stage when the gradient descent algorithm attempts to modify the network weight based on the loss function values for both policy and value networks. During the first stage of experiments, the evaluation of the multi-task learning model was performed on a machine having two cores (dual-core). Under this initial test setup, each worker agent will be running on each individual core, and hence both worker agents are executed in parallel. Now every so often, this global network is going to send its weights to a set of worker agents each with their own copy of policy and value network. Further on each of these individual worker agents will be playing a few episodes of the game under its environment using its network weights from its own experience. From its own experience, each worker agent can calculate its own policy

gradient updates and value updates. Knowledge of these updates will be limited to only these individual worker agents. Eventually, worker agents send their gradient values to the global network so that the global network can update their weights accordingly. Fig. 4.3 is a diagrammatic representation of CNN based model used to implement each of the individual worker agent threads.

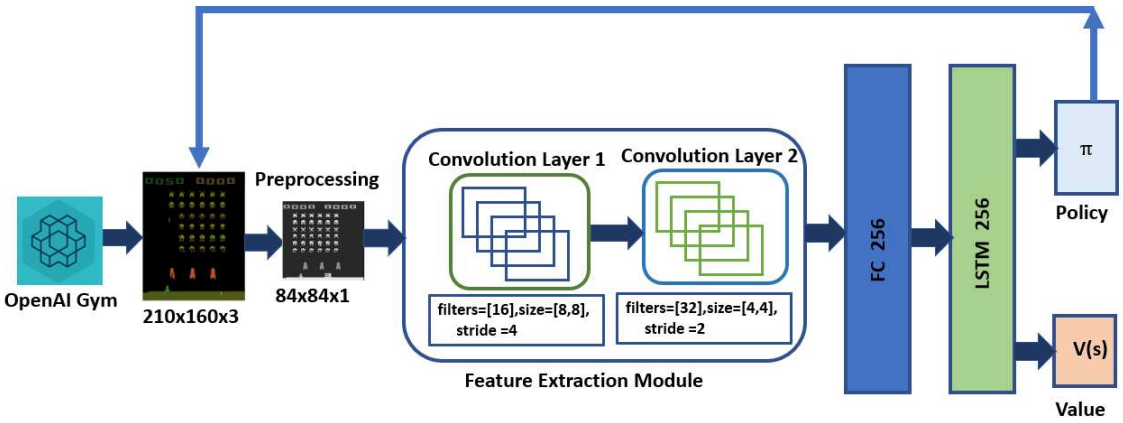

Figure 4.3: CNN based architecture of a single A3C worker agent

Every so often the global network gives its new updated parameters back to its working agents so worker agents are always working with a relatively recent copy of the global network. In this working model, worker threads play episodes of games under their respective environments, find the errors, and calculate the update gradients which will be shared with the global network on a regular basis. Fig. 4.4 shows the sharing of gradient updates by individual worker agents with the global network.

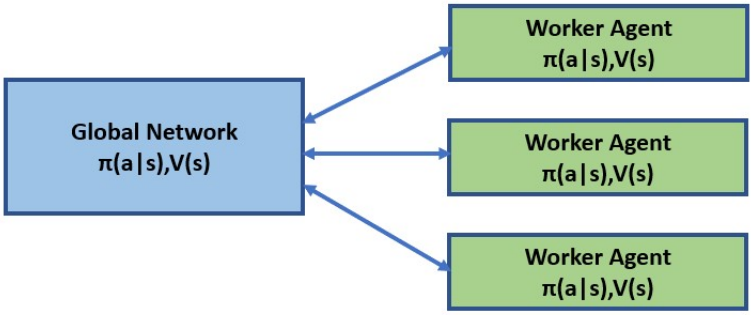

Figure 4.4: Gradient update by worker agents with the global network.

4.5 OpenAI Gym

OpenAI Gym is a toolkit developed by the openAI for the purpose of reinforcement learning research [64]. This toolkit can be used for both developing and comparing reinforcement learning algorithms. Gym supports teaching agents everything from walking to playing games like Pong or Pinball. The gym open-source library provided by the openAI gives you access to a standardized set of environments and entirely compatible with any numerical computation library, such as TensorFlow or Theano. In addition to the gym software library, OpenAI Gym also maintains a website (gym.openai.com) where one could see the scoreboards for all of the environments, showcasing results submitted by various users. OpenAI Gym has been designed based on the idea of the episodic setting of reinforcement learning, wherein an agent's experience is broken down into a series of episodes. During each episode play, the agent's initial state is randomly chosen or sampled from a distribution, and further on its interaction proceeds until the environment reaches a terminal state for the specific game. The objective or goal during episodic reinforcement learning is to maximize the expectation of total reward per episode. With this, an agent

aims to achieve a high level of performance in as few episodes as possible. In our current experiments, we have employed the hybrid multi-task model to function within the Atari 2600 gaming environment with the help of the Gym library.

4.6 TensorFlow

TensorFlow is an end-to-end open-source platform developed by Google for a machine learning system. It is designed to operate at a large scale and in heterogeneous environments. The TensorFlow framework uses dataflow graphs to represent computation, shared state, and the operations that change the state [65]. Within TensorFlow computations are expressed as stateful dataflow graphs, which could contain nodes across different machines within a cluster, and also within a machine across multiple computing devices such as multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs) [66]. TensorFlow provides support for a variety of applications, with a focus on training and inference on deep neural networks. In our experiments conducted with a hybrid multi-task model, we have used the Tensorflow_gpu-2.0.0 version which supports the operation of TensorFlow-based networks on GPUs. It also supports the stable operation of Python-based applications [67].

4.7 PyCharm

PyCharm is an integrated development environment (IDE) used specifically for the Python language. As an IDE, it provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and data science with Anaconda [68]. For the development of the hybrid multi-task learning model, we have used Windows-based PyCharm Community Edition that is released under the Apache License and used Python 3.7 version.

Table 4.1. Summary of Development Environment.

Name	Details
OS	Windows 10
IDE	PyCharm 2019.3.5
Language	Python 3.7
Machine Learning Library	TensorFlow GPU-2.0.0
NVIDIA GPU Library	CUDA Toolkit 10.0
NVIDIA GPU Library	cuDNN v7.6.5
NVIDIA Deep Learning Library	TensorRT 6.0
NVIDIA HPC GPU	Quadro P5000
Cloud Server OS	Windows 10
Reinforcement Learning Toolkit	OpenAI Gym 0.18.0

4.8 Summary

This chapter has detailed the implementation of the hybrid multi-task learning model including details of the tools and libraries used. Details of the implementation have provided the information of the multi-task worker agent model, and how the multi-task worker agents will be used for the multi-task learning within the Atari 2600 based gaming environment. In addition, details related to tools and libraries such as PyCharm, TensorFlow, and OpenAI Gym are also provided. In the next chapter, the evaluation of the implemented hybrid multi-task learning model and related results obtained are presented. In addition to this, there would be details related to the analysis of the test results obtained during the experiments with the hybrid multi-task learning model.

Chapter 5

Evaluation and Results

This chapter presents the details on the experiments conducted with the proposed hybrid multi-task learning model on Atari 2600 based gaming environments under multiple test configurations and associated results obtained. Initial sections of this chapter focus entirely on the experiment efforts done with both desktop-based test environment as well as cloud-based environment namely Paperspace and present the results collected. Following this, the latter part of the chapter focuses on analyzing both the test results obtained as well as how well a hybrid multi-task learning model could optimize the performance of DRL agents by mitigating the DRL challenges.

5.1 Model Validation Methodology

This section provides details on the evaluation of the hybrid multi-task learning model and the associated results obtained during those experiments. Having said that hybrid multi-task model learning attempts to optimize the performance of a DRL agent by hybrid multi-task learning approach, it leverages the A3C algorithm's multi-threaded and asynchronous approach to deep reinforcement learning [69]. This algorithm gives the capability to have a model to be trained with multiple, different explorations of a single target task, providing data sparsity, and avoiding the use of memory replay [5]. Given the multi-threading characteristics, the proposed hybrid model attempts to leverage A3C's ability to perform multi-task learning without modifications when applied to different, but semantically related tasks. To do so, we simultaneously train multiple tasks using a single A3C model, allowing the network to asynchronously share knowledge obtained from and to all tasks.

The hybrid A3C model attempts to learn two different tasks and then combine the learning to accelerate the performance. Evaluation of the proposed hybrid multi-task model will be conducted on a prototype based on the A3C model and trained with the Atari 2600 environment provided in the OpenAI Gym. Validation of the proposed hybrid multi-task model is done by using RL environments based on Atari 2600 games, and the various games used within the experiments are from the Atari-57 family. All the games from this family are designed with the same resolution parameters (210x160x3) such that the proposed hybrid multi-task framework could be tested with any other game from this family without needing any change in the design. The feature extraction module within the hybrid multi-task learning model is compatible across all the games from Atari-57. The decision to choose Atri2600 games for the validation of the model was primarily due to the availability of the OpenAI Gym library that provides the APIs game creation with a built-in reward structure. It is also possible to extend this model validation with other real-time environments such as autonomous driving, and in such a case the feature extraction module needed to be modified as the resolution of the images could be much higher. This in turn means that there will be changes in terms of preprocessing image resolution, number of convolution layers, number of filters, size of the filters, type of the filter, filter stride values. In addition to this, the developer should be also taking care of the DRL agent's reward structure for that environment. A3C algorithm used for the experiments will be based on Google DeepMind's paper titled-asynchronous methods for deep reinforcement learning.

5.1.1 Single Agent Actor and Single Agent Critic

As a preliminary step towards the development of the proposed system model, the initial set of experiments are conducted on a desktop-based environment with the game of

CartPole-v0 which is having a finite set of action and state space. The methodology followed was to individually develop the single-agent actor which is based on policy gradient, and similarly a single agent critic which is a value-based network to measure the performance. Both these networks were developed as the standard feedforward neural networks and experiments are conducted for the finite number of episodes. As an outcome of the experiment performance of both single-agent actor and critic are measured.

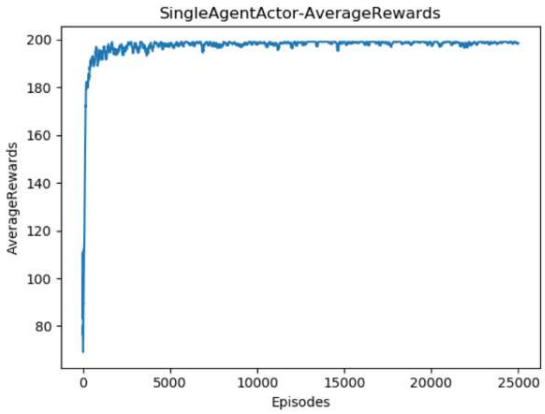

Figure 5.1: Single-agent actor – average rewards.

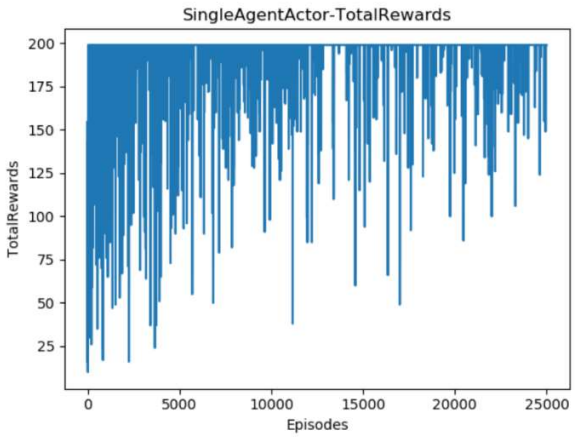

Figure 5.2: Single-agent actor- total rewards.

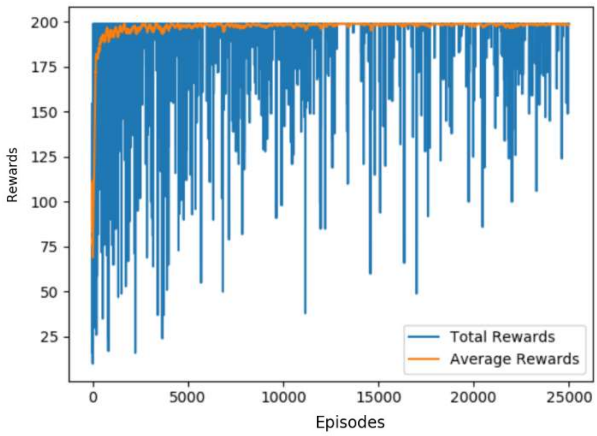

Figure 5.3: Single-agent actor- merged results.

Fig. 5.1 to Fig. 5.3 represent the test results generated for the single-agent actor model based on the feedforward neural networks model, and Fig. 5.4 to Fig. 5.6 represent the test results generated for the single-agent critic model based on the feedforward neural networks.

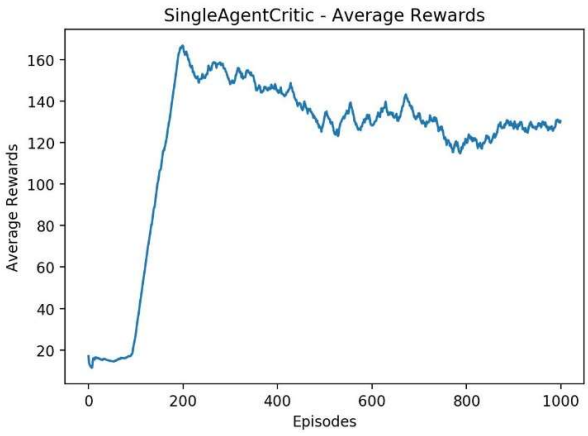

Figure 5.4: Single-agent critic-average rewards.

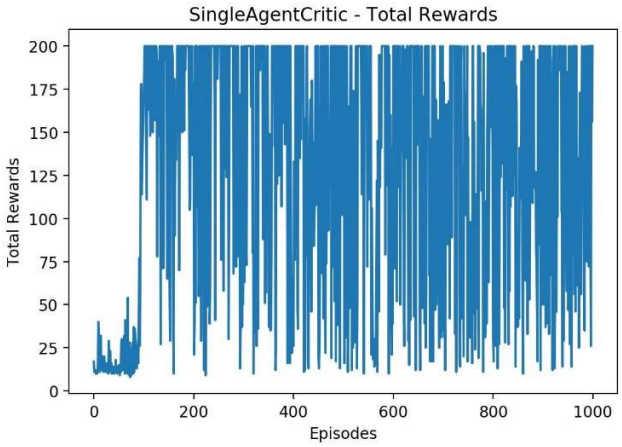

Figure 5.5: Single-agent critic-total rewards.

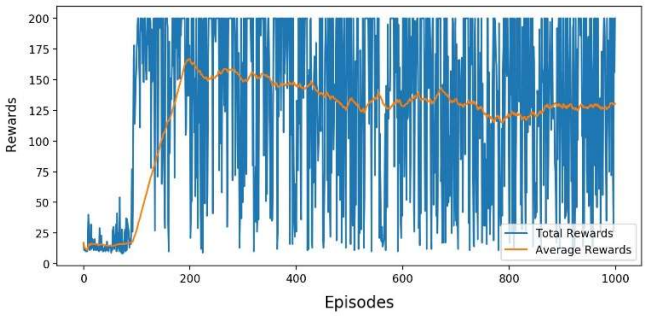

Figure 5.6: Single-agent critic-merged rewards.

It is evident from the statistics that policy gradient-based actor is able to increase the rewards over the episodes gradually. At the same time, the value-based critic module is able to show the increment in performance in the early episodes, with a small dip in the mid episodes with a fluctuating result for the forthcoming episodes.

5.1.2 A3C Multi-Worker Model on Desktop Platform

At the initial stage of the evaluation of the proposed hybrid multi-task learning model, A3C based multi-worker model is built and verified with a single OpenAI Atari 2600 gaming environment. During the course of the experiments, multiple OpenAI Atari 2600 gaming environments will be used for the evaluation of the proposed model, which involves Pong-

v0, Breakout-v0, SpaceInvaders-v0, DemonAttack-v0, and Pheonix-v0. During the first stage of evaluation, the performance of the reinforcement learning agent will be measured individually on each of these gaming environments to generate the initial test statistics. The test results generated by the A3C model were trained within the OpenAI Atari 2600 environments provided in the OpenAI Gym [20]. In the next step towards the evaluation of a proposed hybrid multi-task model, the A3C algorithm based on a multi-worker agent-based environment is created for a more graphic intense Atari 2600 game environment Breakout-v0 as represented by Fig 5.7. From the perspective of a DRL agent, this environment is being treated as a complex one as we will be having an infinite number of state-action spaces to deal with. In order to accommodate and handle this environment, a convolutional neural network (CNN) based model was used for the validation. This configuration was tested under a desktop-based environment by using a multi-task environment having four worker threads that combinedly executed 500,000 steps of the game. Each of the individual threads is having its own individual copy of the environment but different from one another in terms of the view of the gaming environment. The proposed hybrid multi-task learning model brings the aspect of optimization of DRL agents by having multi-agent-based multi-task learning between the two semantically similar environments. This model helps to achieve the DRL agent optimization objective by sharing the knowledge (gradients) by multiple - workers running simultaneously within the hybrid environments. Due to the semantic similarity between the tasks running in hybrid environments, the revised parameters shared by the global network with individual workers would help them to use the consolidated knowledge in optimizing their individual policies.

This in turn would lead to taking better actions at each stage and thereby increase the expected sum of future rewards.

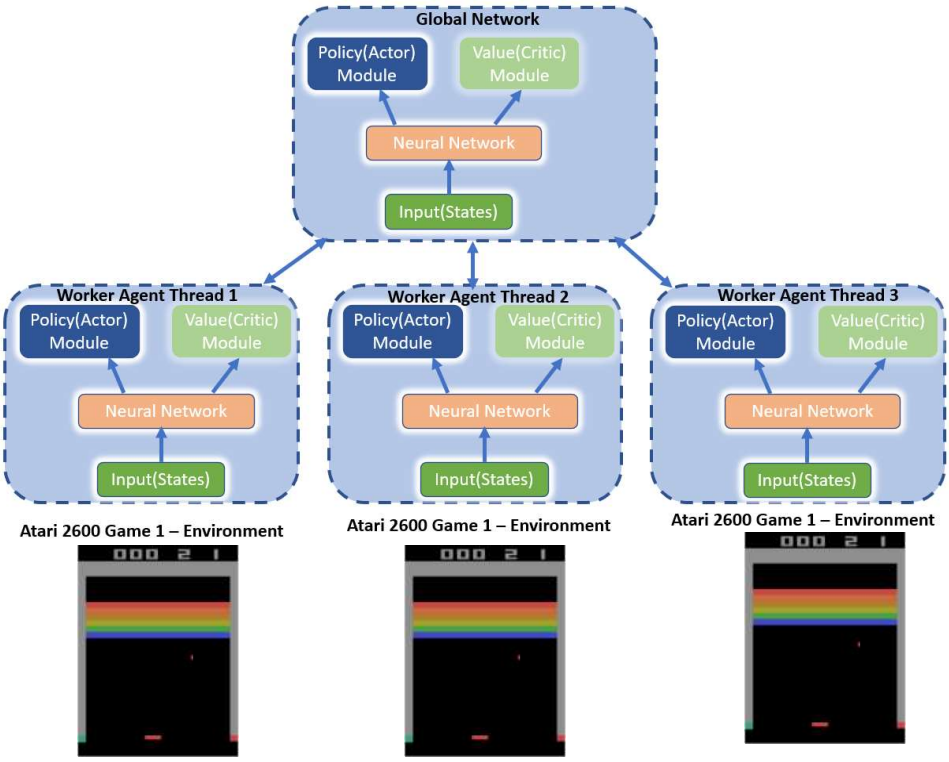

Figure 5.7: A3C multi-task workers environment for Breakout-v0.

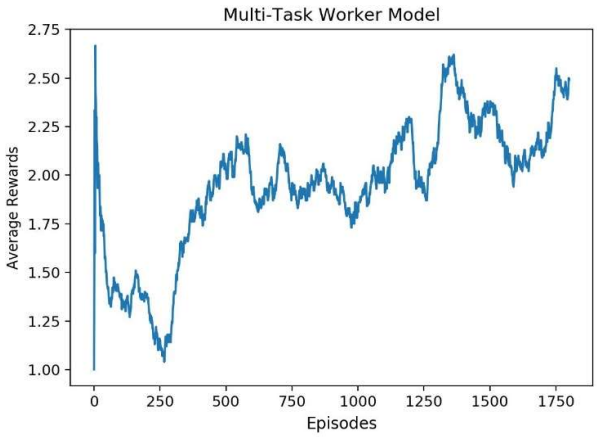

Figure 5.8: Breakout-v0 multi-task workers model-average rewards.

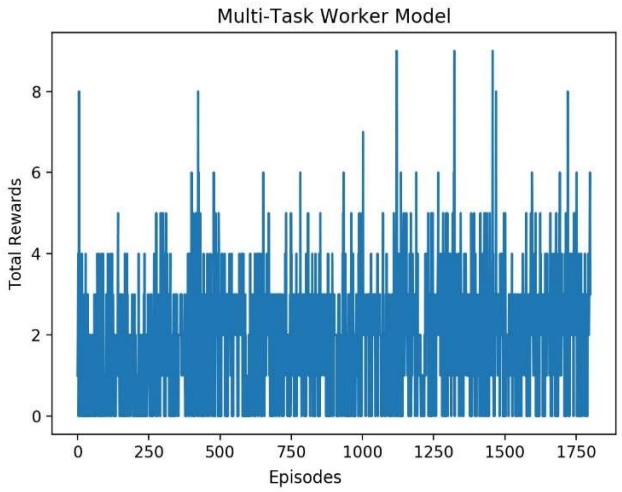

Figure 5.9: Breakout-v0 multi-task workers model-total rewards.

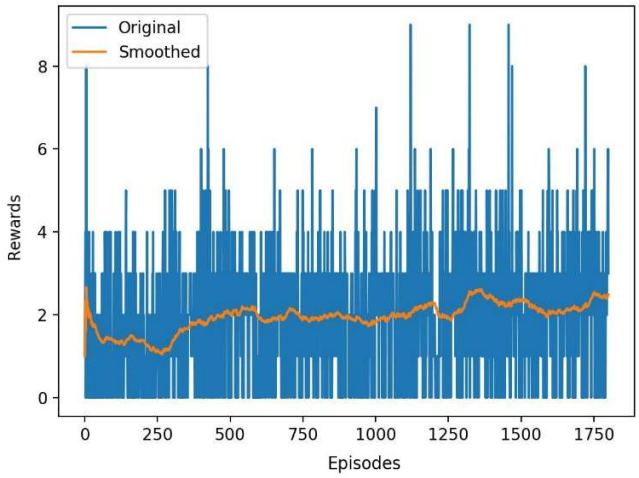

Figure 5.10: Breakout-v0 multi-task workers model – merged results.

Fig. 5.8 to Fig. 5.10 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Breakout-v0. This testing was carried out by using 4-worker agents or worker threads based A3C model to generate the initial set of results of a desktop-based test environment. As a further attempt towards the evaluation of the proposed hybrid multi-task model, the A3C algorithm-based

multi-worker agent environment is also created for one more graphic intense Atari 2600 game environment named- Pong-v0. The decision to choose Pong-V0 was after the careful examination of the high level of similarity level among these two games, Breakout-V0 and Pong-V0. Having a reasonable level of similarity could act as an accelerator during the validation of the proposed hybrid multi-task learning model execution. Similar to the way how Breakout-v0 was tested earlier under a multi-task worker environment, the Pong-v0 game was also tested under a desktop-based environment by using a multi-task environment having four worker threads that combinedly executed 5 million steps of the game. Each of the individual threads is having its own individual copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent. In order to accommodate and handle the Pong-v0 gaming environment, a similar CNN-based model was used during the validation of the multi-task learning model. In both Pong and Breakout, a player must control a paddle in order to hit a ball. For Pong, the player must attempt to make an opponent miss the ball, while for Breakout the goal is to break as many bricks as possible. Fig. 5.11 shows the snapshot view of the Breakout-v0 and Pong-v0 environments.

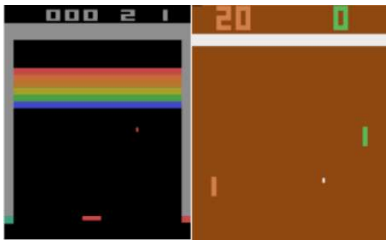

Figure 5.11: Snapshot of Breakout-V0 and Pong-v0 game environment.

Fig. 5.12 to Fig. 5.14 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Pong-v0.

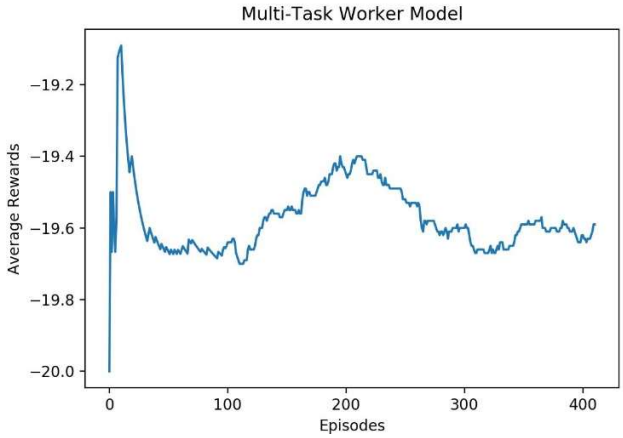

Figure 5.12: Pong-v0 multi-task workers model-average rewards.

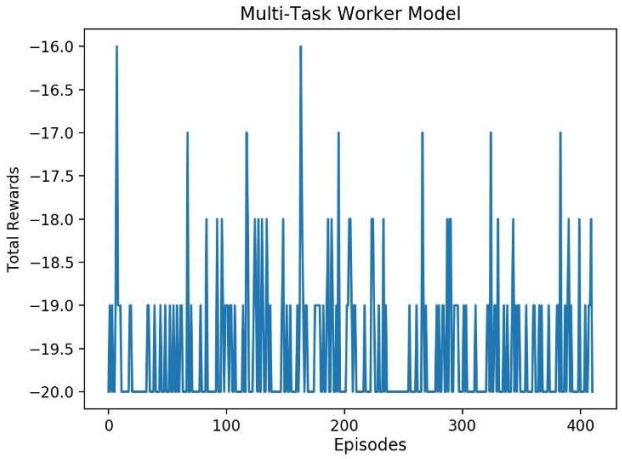

Figure 5.13: Pong-v0 multi-task workers model-total rewards.

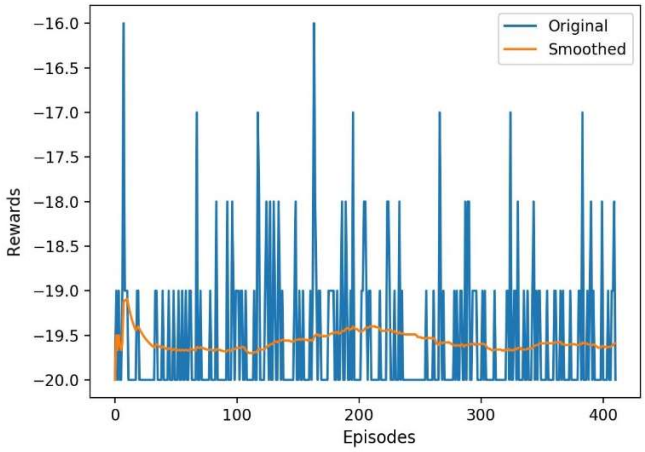

Figure 5.14: Pong-v0 multi-task workers model-merged results.

As part of the detailed and exclusive evaluation of the proposed hybrid multi-task model, we decided to pick one more pair of Atari 2600 games namely Space Invaders-v0 and DemonAttack-v0 from the Gym library. The decision to choose these two games as the second test pair was after the examination of the high level of semantic similarity between their pattern play. Both these games are based on the theme of shooting wherein the player should be able to control a moving ship with the capability of shooting and hitting the enemies. The following Fig. 5.15 shows the snapshot view of the SpaceInvader-v0 and DemonAttack-v0 environments.

Figure 5.15: SpaceInvaders-v0 and DemonAttack-V0 environment.

In terms of complexity, Space Invaders is relatively less complex as the enemies in this game move more in a regular fashion than in the other game. Whereas in Demon Attack, there are a wide variety of enemies who moves more randomly with the capability to shoot back, which makes the gameplay more complex from the perspective of the RL agent. More importantly, every game used in this experiment has its own reward structure that is in-built by the Gym library. In other words, even though there is some level of semantic similarity between the games chosen within each test pair, the scoring and reward structure followed within each game is unique.

Similar to the way how previous two games from the first pair were tested, the SpaceInvaders-v0 game was also tested under a desktop-based test environment by using a multi-task worker model having four worker threads that combinedly executed about 500,000 steps of the game. Fig. 5.16 to Fig. 5.18 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Space Invaders-v0.

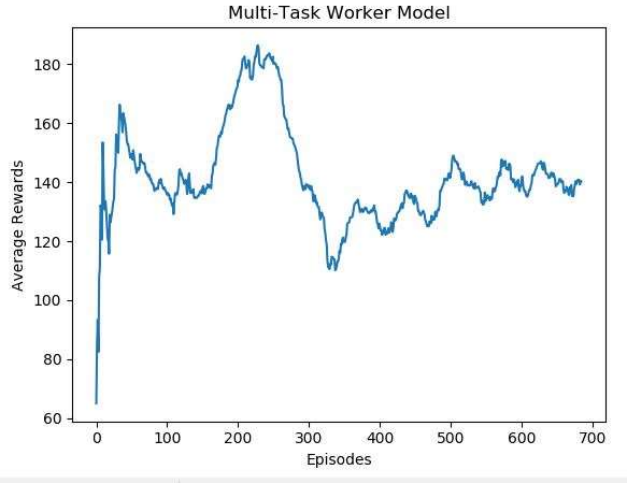

Figure 5.16: SpaceInvaders-v0 multi-task workers model-average rewards.

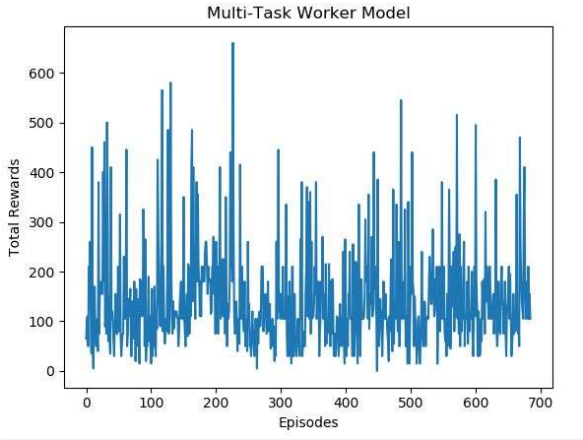

Figure 5.17: SpaceInvaders-v0 multi-task workers model-total rewards.

Each of the individual threads is having its own individual copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent. In order to accommodate and handle the Space Invader-v0 gaming environment, a similar CNN-based model was used during the validation of the multi-task learning model. This testing was carried out by using a 4-worker agents-based A3C model to generate the initial set of results of a desktop-based test environment.

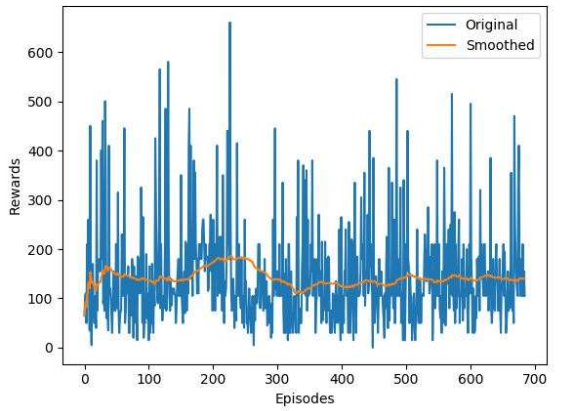

Figure 5.18: SpaceInvaders-v0 multi-task workers model- merged results.

Similar to the way how Space Invader-v0 was tested earlier under a multi-task worker environment, the DemonAttack-v0 game was also tested under a desktop-based environment by using a multi-task environment having four worker threads that combinedly executed 500,000 steps of the game.

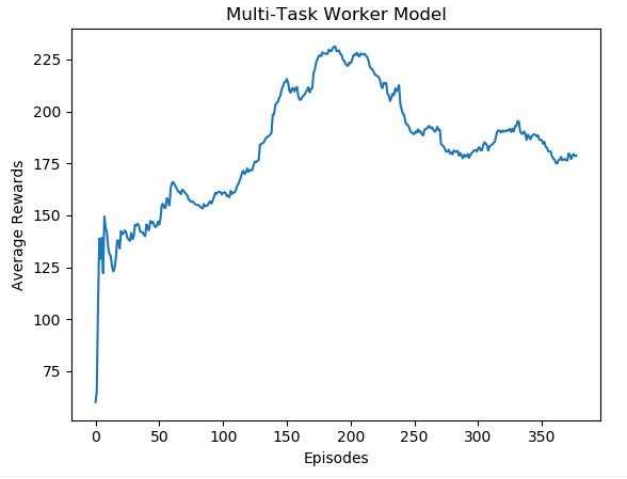

Figure 5.19: Demon Attack-v0 multi-task workers model-average rewards.

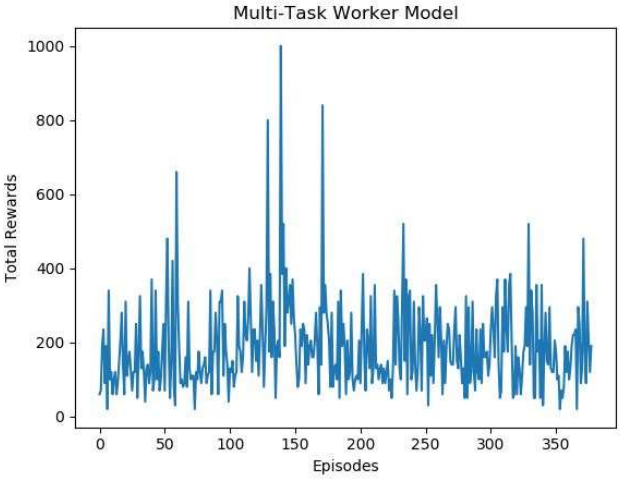

Figure 5.20: DemonAttack-v0 multi-task workers model-total rewards.

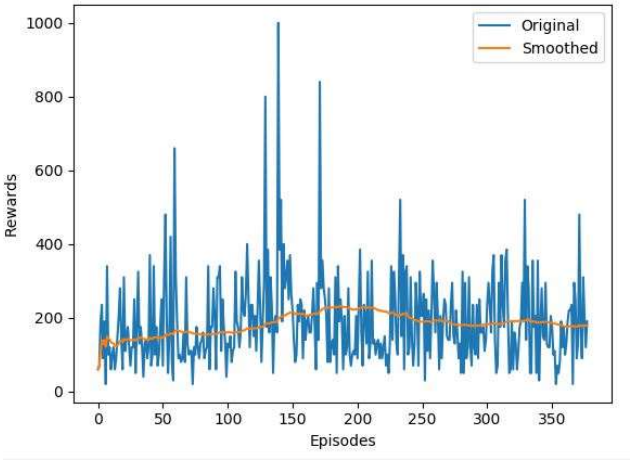

Figure 5.21: DemonAttack-v0 multi-task workers model-merged results.

Each of the individual threads is having its own individual copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent. Fig. 5.19 to Fig. 5.21 shows the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Demon Attack-v0.

5.1.3 A3C Multi-Worker Model on Paperspace Cloud Platform

In order to test and generate better results with a higher number of episodes of gameplay for each game under the proposed hybrid multi-task model, we decided to test the proposed model under a cloud-based test environment. As part of this, we opted to move our testing to machines with GPU with CUDA cores support under the cloud environment hosted by Paperspace [70]. This allowed us to rent a server in the cloud with much higher throughput than that of our local machine. Paperspace server used has up to 8GB of graphic memory and 32 GB of RAM and equipped with NVIDIA GPU - Quadro P5000 having CUDA

support (with 2560 CUDA cores) to facilitate the parallel computing for deep learning applications [71]. During this process, we configured a couple of Windows OS-based virtual test machines namely Gen 2 (P4000) having NVIDIA GPU supported with CUDA cores in the cloud environment.

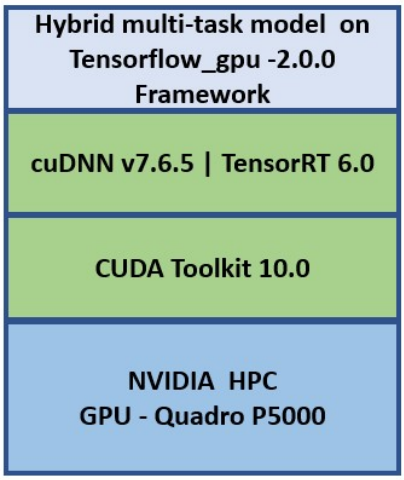

Figure 5.22: Test environment of Paperspace cloud server machine.

Each of the Atari2600 games was tested with 8 worker agents for a higher number of global steps. In order to capture the test results, a tensor board visualization tool was employed which uses the event file captured during the test execution to generate the test execution results.

Fig. 5.23 to Fig. 5.26 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environments under the virtual test machines under the cloud environment. Note that test result figures on the Paperspace cloud server environment were generated within Tensor Board (TensorFlow's visualization toolkit). In all those figures the numbers on the x-axis represent the global steps in millions

(taken by the agent), and the numbers on the y-axis represent the rewards (game score). The same convention applies to Figures 5.23 to 5.39.

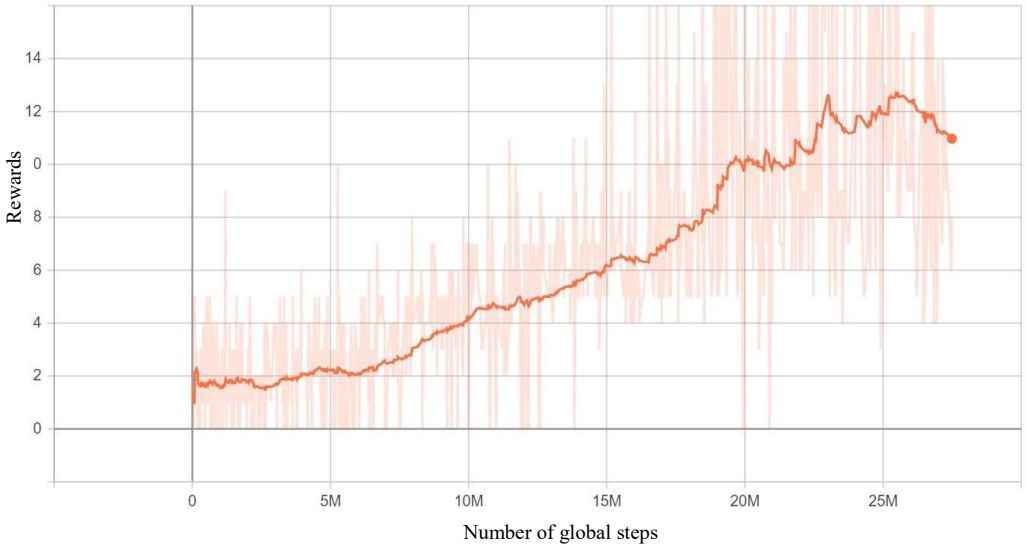

Figure 5.23: Breakout-v0 standalone test result with 8 multi-workers.

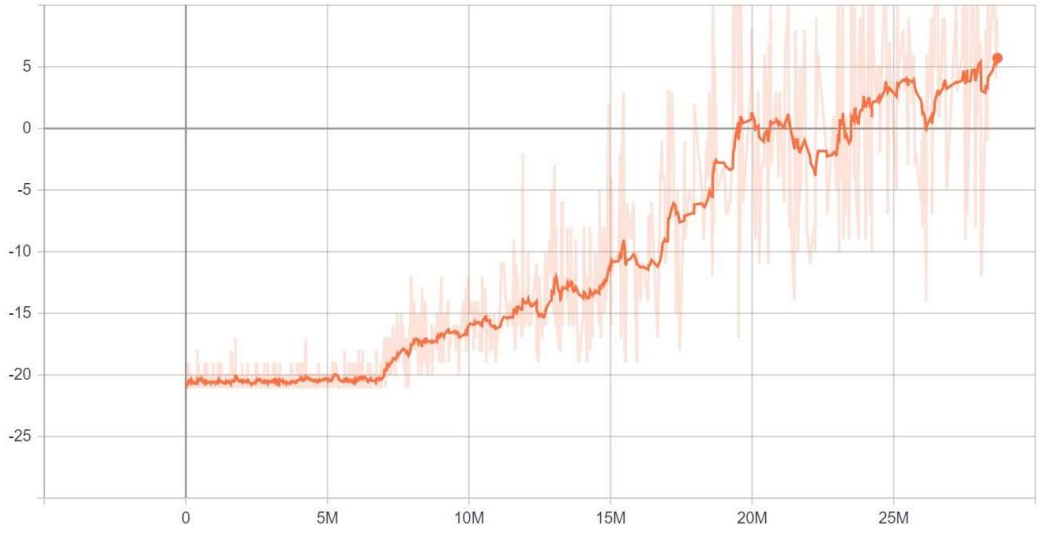

Figure 5.24: Pong-v0 standalone test result with 8-multi-workers.

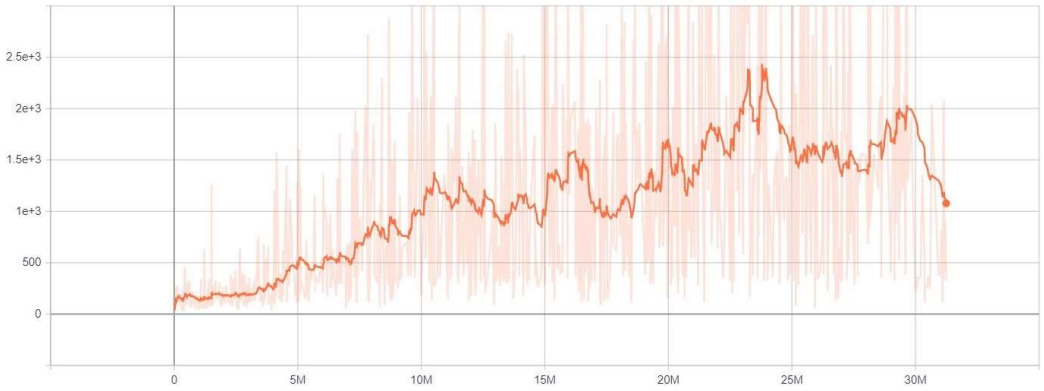

Figure 5.25: DemonAttack-v0 with 8-multi-task workers.

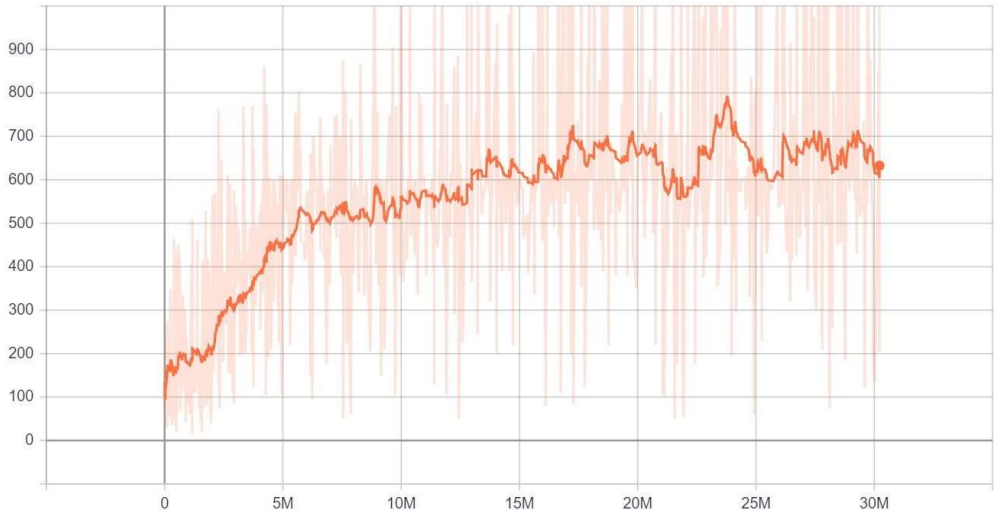

Figure 5.26: SpaceInvaders-v0 with 8-multi-task workers.

5.1.4 Hybrid Multi-Task Learning Model

Now, as the next step in the verification of our proposed hybrid multi-task model, we have tested the model by running two semantically similar games simultaneously.

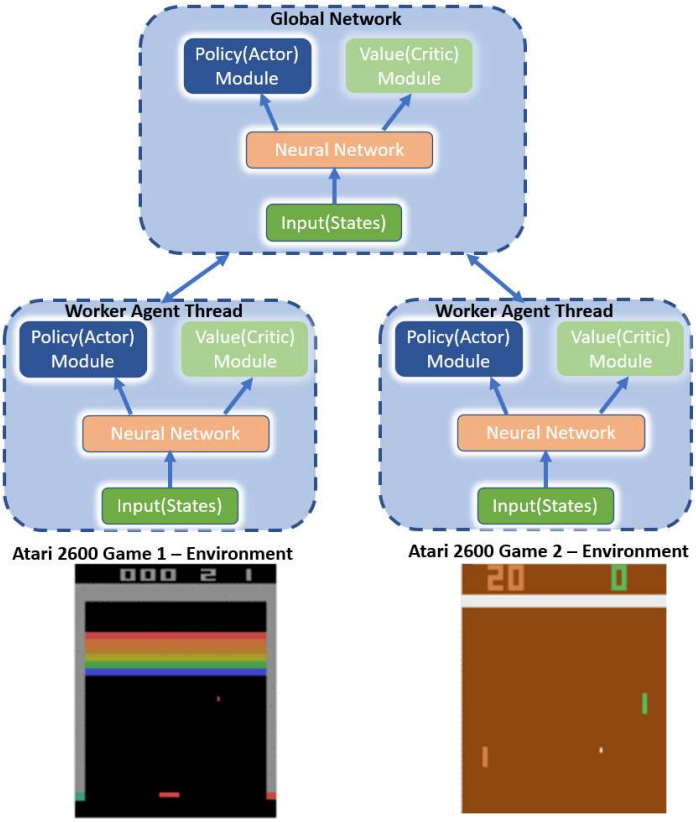

Figure 5.27: A hybrid multi-task model of Breakout-v0 and Pong-v0.

At the end of testing, the individual test score for each game was captured. Since we have chosen two pairs of games with semantic similarity, we created a separate test setup for each pair. Fig. 5.27 shows the diagrammatic representation for each pair under the hybrid multi-task model. In order to maintain the uniformity of testing, each of the individual games was tested with 8 worker agents which totals to 16 worker threads altogether within the test environment. Fig. 5.28 and Fig. 5.29 respectively show the test execution results captured for breakout-v0 and Pong-v0 under the joint test environment.

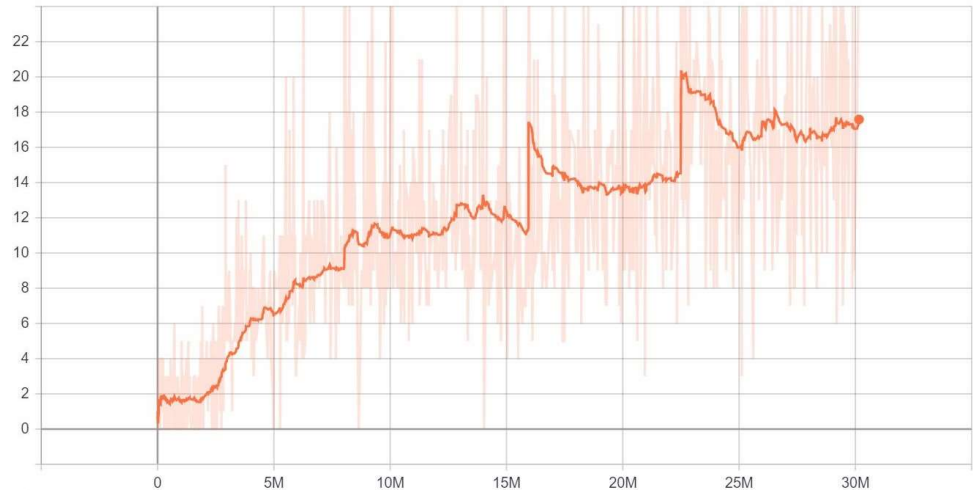

Figure 5.28: Breakout-v0 test results with the hybrid multi-task model.

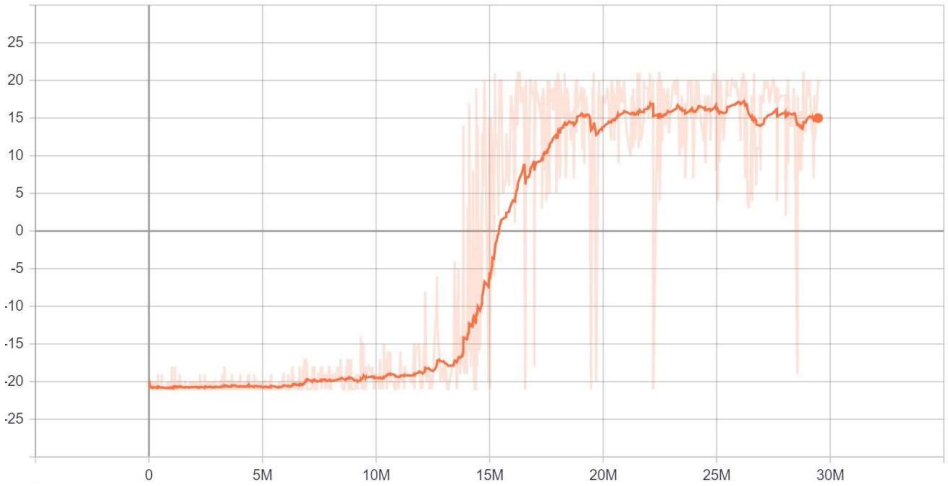

Figure 5.29: Pong-v0 test results with the hybrid multi-task model.

These test results are generated based on the experiments conducted with the Paperspace cloud server machines having the Nvidia GPU supported by CUDA cores. This environment facilitates the large-scale testing for the hybrid multi-task model having a CNN-based feature extraction module. In a similar fashion, we created the joint test environment for the second test pair consisting of Atari2600 gaming environments, SpaceInvader-v0, and DemonAttack-v0. In order to maintain the uniformity of testing,

each of the individual games was tested with 8 worker agents which totals to 16 worker threads altogether within the test environment.

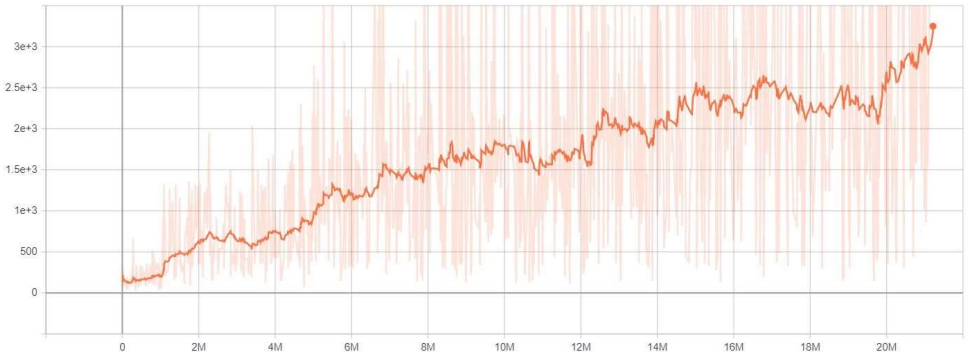

Figure 5.30: DemonAttack-v0 test results with the hybrid multi-task model.

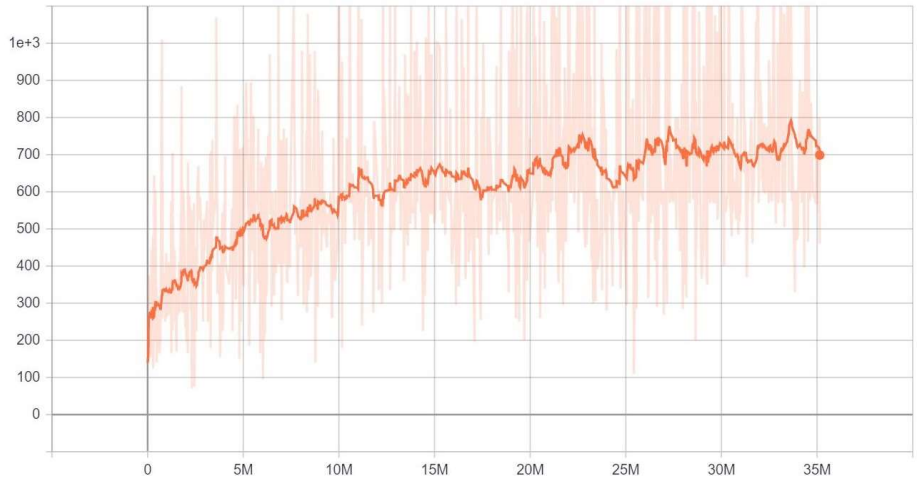

Figure 5.31: SpaceInvaders-v0 test results with the hybrid multi-task model.

Fig. 5.30 and Fig. 5.31 show the test results for each of the individual games within the tested pair of games. In order to measure the impact of the DRL agent's performance under the proposed hybrid multi-task model while testing with environments with high semantic dissimilarity, we have also conducted two pairs of testing. In this testing first pair

of testing was done using DemonAttack-v0 and Pong-v0, which are having a high level of semantic dis-similarity level. Under this test environment, the performance of each of the individual games will be measured to see the impact of negative knowledge (gradient transfer). A similar test setup will be made ready for the second pair consisting of Atari2600 gaming environments namely, SpaceInvader-v0 and Breakout-v0. Fig. 5.32 and Fig. 5.35 show the diagrammatic representation for each of the test pairs mentioned above.

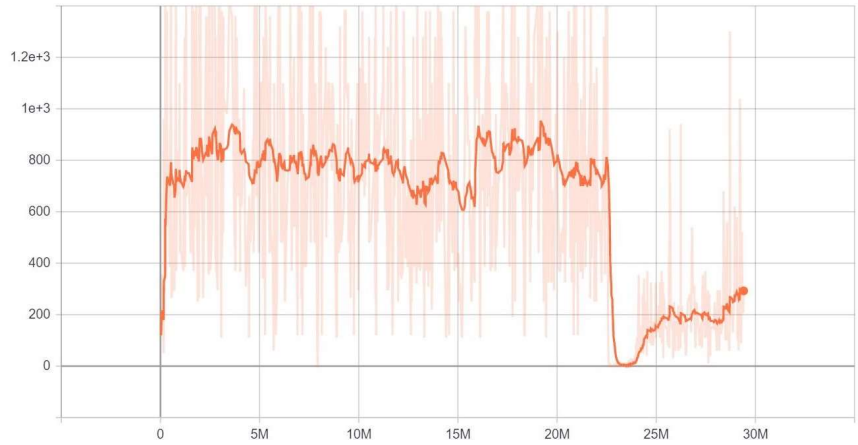

Figure 5.32: DemonAttack-v0 results for the semantic dissimilar test.

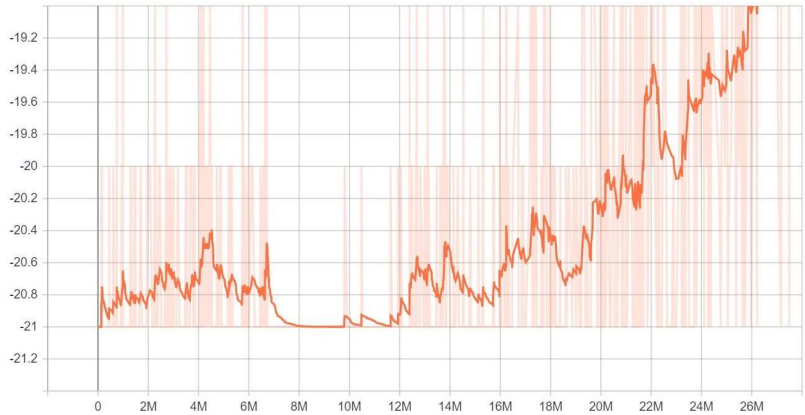

Figure 5.33: Pong-v0 results for the semantic dissimilar test.

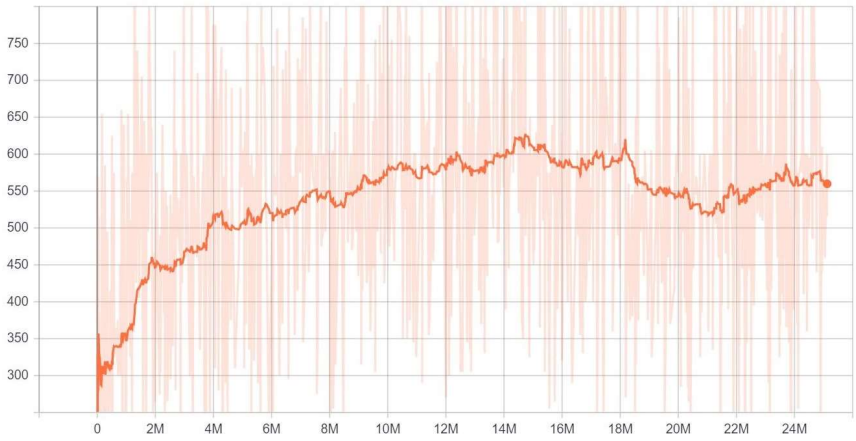

Figure 5.34: SpaceInvaders-v0 results for the semantic dissimilar test.

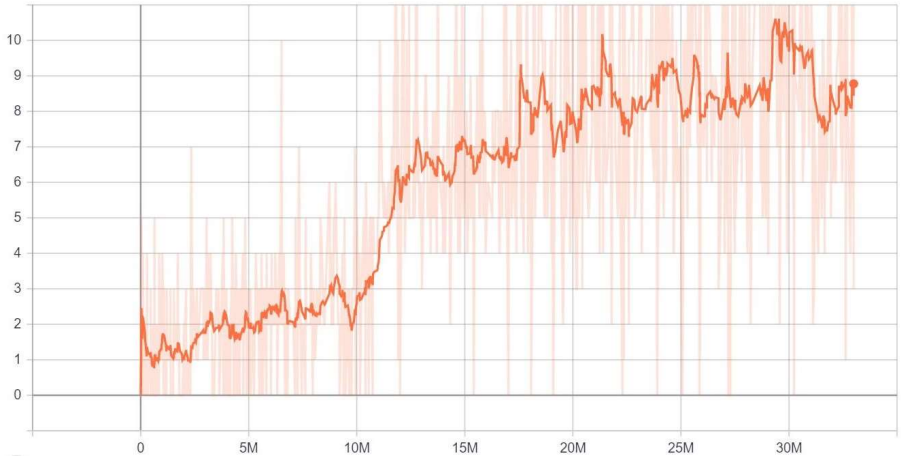

Figure 5.35: Breakout-v0 results for the semantic dissimilar test.

During our test efforts, we also conducted experiments to measure the impact of individual game scores when the hybrid multi-task model is tested with three semantic similar games namely SpaceInvader-v0, DemonAttack-v0, and Pheonix-v0. Fig. 5.36 shows the hybrid multi-task learning model for the same configuration. Even though each of these games has a semantic similarity factor, at the same time, each of them is having its own reward structure.

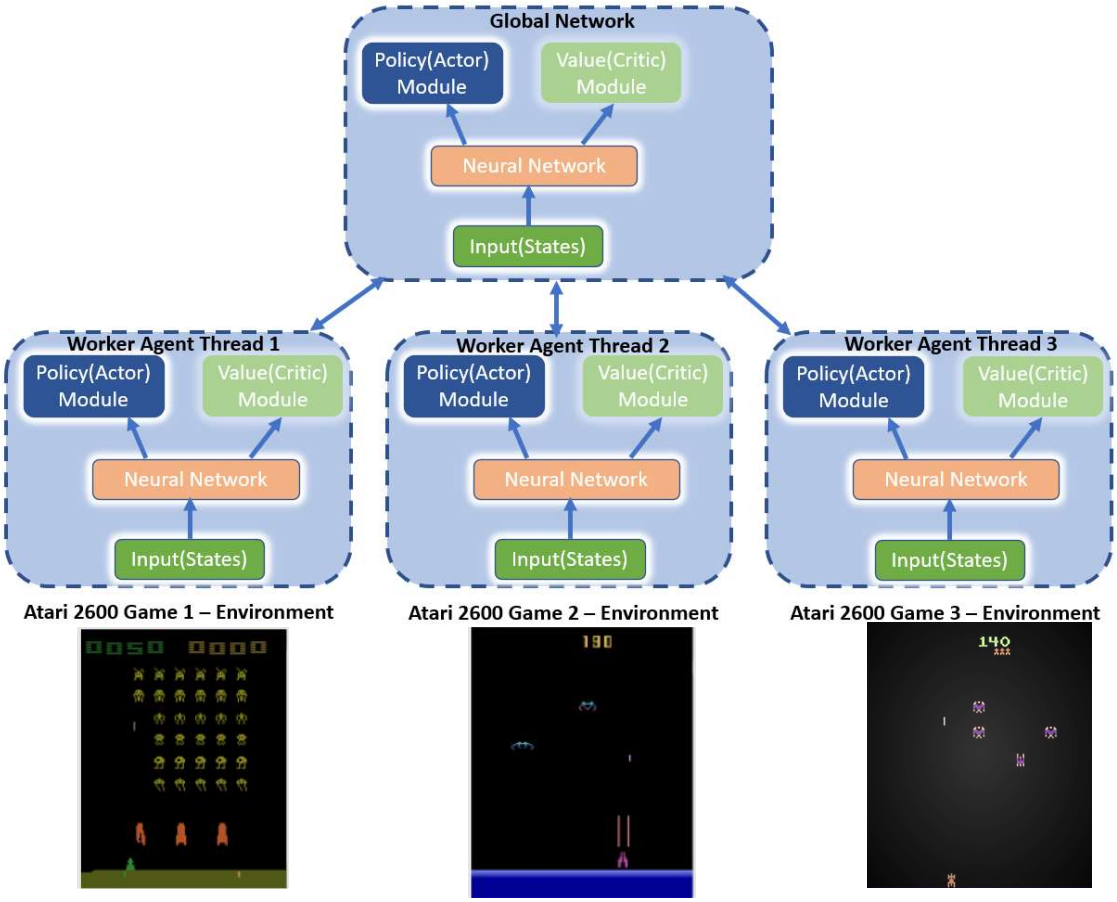

Figure 5.36: The hybrid multi-task model of SpaveInvader-v0, DemonAttack-v0, and Pheonix-v0.

Fig. 5.37 to Fig. 5.39 show the respective test results captured with the hybrid multi-task model for the three OpenAI Atari 2600 gaming environments with a high level of semantic similarity

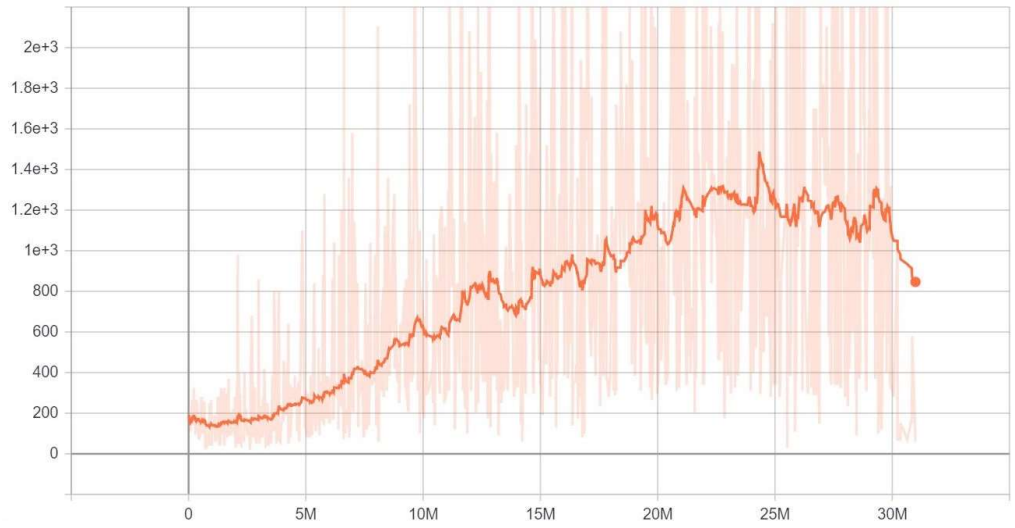

Figure 5.37: DemonAttack-v0 test results with hybrid multi-task model for 3 semantically similar environments.

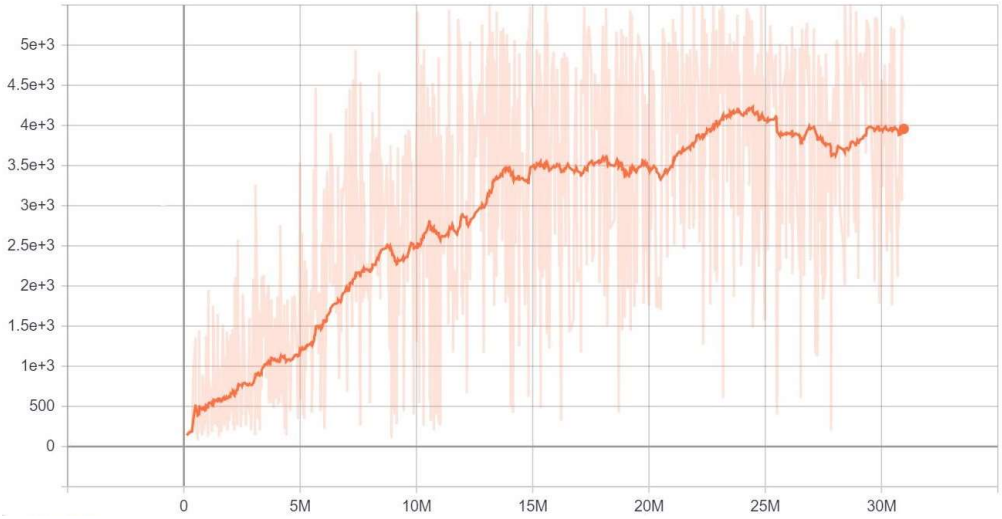

Figure 5.38: Pheonix-v0 test results with hybrid multi-task model for 3 semantically similar environments.

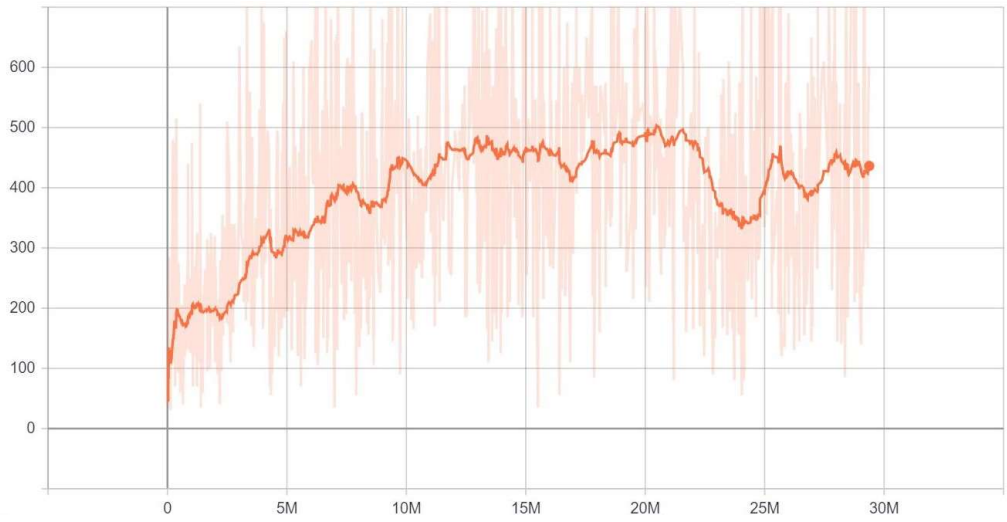

Figure 5.39: SpaceInvaders-v0 test results with hybrid multi-task model for 3 semantically similar environments.

5.1.5 Summary

The details mentioned under the subsections of section 5.1 have presented the evaluation of the hybrid multi-task learning model by using the Atri2600 gaming environment. The model was tested with two pairs of games, with each pair having individual games with a high level of semantic similarity to measure the impact of the hybrid multi-task learning model in optimizing the performance of the DRL agent. Likewise, experiments were also conducted to measure the impact of negative knowledge transfer on the model by running the experiments on pairs of games having a high level of dissimilarity. Additionally, the model was also used to test the impact on performance when a greater number of Atari 2600 games with semantic similarity are executed together with the hybrid multi-task learning model.

5.2 Analysis of Test Results

This section analyzes the test results obtained with the hybrid multi-task model tested with various Atari 2600 gaming environments. In the first stage of the testing, we conducted a standalone kind of testing with each of the individual gaming environments individually. In order to conduct this testing, we have created the A3C algorithm-based multi-thread model wherein each of the games is tested by using 8 worker threads. In order to maintain the uniformity of the testing throughout this experiment, we have kept the count of worker threads as 8 for all the gaming environments. We tested our model by adding the final LSTM layer after the feedforward network to obtain the best performance of the A3C algorithm as a whole. We have extensively used NVIDIA GPU - Quadro P5000 having CUDA support (with 2560 CUDA cores) to facilitate parallel computing as it involves the use of CNN to process game screen images. More importantly, in the first stage of testing, we choose two sets of games, with set 1 consisting of Breakout-v0 and Pong-v0, then set 2 consisting of games SpaceInvaders-v0 and DemonAttack-v0. The decision to choose these games to form two sets was after the clear examination of semantic similarity factor among them. As anticipated, the base A3C-based multi-thread model was able to achieve performance enhancement on all of these games during the testing due to the parallel multi-task learning aspect of A3C. We have conducted the testing for 25 million to 30 million global steps for each of these individual games to have convincing test results for comparison with future state tastings planned [72].

In the second stage of testing, we experimented with our proposed hybrid multi-task model approach, wherein we trained two games, but with high-level semantic similarity, simultaneously. In contrast to the first stage testing, where gradients shared to

the global network by worker agents are all of the same types, in the hybrid environment we have two different types of worker threads. As it is anticipated, the performance of individual games under the hybrid environment was not on par with standalone performance results obtained with the first stage of testing. As and when the progress of the game, we could see the impact of positive knowledge sharing among these two tasks that are trained jointly. Due to the semantic similarity among them, updates shared by the global network could mitigate some of the key challenges associated with partial observability in comparison to a single game-based environment. Based on the test results obtained with each of the sets that we mentioned earlier, we could see that each of the games under each set could boost its performance over the course of the training. By this, we can establish that our hybrid multi-task model is able to learn multiple similar gaming tasks simultaneously without degradation in performance for any one of the individual gaming tasks [72]. In comparison to the state-of-the-art methods discussed which are based on the distillation methodology, the hybrid multi-task model adopts to train and learn the method for a multi-task actor-critic network from the scratch. Along with this, the hybrid multi-task approach also measures the impact amount of positive knowledge transfer done through parameter sharing. As we have adopted a model-free-based approach, it is relatively less computationally intensive compared to a model-based approach.

In the next stage of testing with the hybrid multi-task model, we conducted experiments by testing the hybrid multi-task model with two different pairs of games with a high level of semantic dissimilarity. As we could see from the test results obtained, negative knowledge transfer or the gradients shared by two semantically dissimilar worker training threads had a huge impact on the individual games' score. As the test results

indicate, all the individual games 'performance was hugely affected due to negative knowledge transfer. Finally, we also tested our model to see the impact on the positive knowledge transfer by training more than two semantically similar tasks with the same number of workers allocated to each game [72]. The test results obtained indicate that as the number of worker threads increases, updates shared by the global network deteriorates in comparison to a hybrid multi-task model with two semantically similar tasks. This situation possibly requires more tuning on the hyperparameter front as well as catastrophic forgetting of the neural networks of the gaming environments, which will be addressed in the future work planned.

The objective behind the proposed hybrid multi-task learning model is to leverage multi-task learning capabilities offered by the core actor-critic methodology by using the A3C algorithm to optimize the DRL's performance. By having a hybrid multi-task-based learning environment, wherein agents belonging to different but semantically similar games, we aimed at addressing some of the key challenges associated with the existing multi-task DRL. In order to showcase, the extent to which our model could address those issues, we would like to have a case study based on the test results obtained. For this purpose, we are using both the standalone and hybrid model test results obtained for the Breakout-v0 game as indicated by Fig. 5.23 and Fig. 5.28 respectively. In order to have a fair comparison and derive a convincing conclusion, we have ensured that the same amount of resources have been allotted in both test scenarios in terms of the number of worker threads, test configurations, and the number of global steps taken parameter. By having a comparison of these two test results, it is quite evident that in terms of the training time needed, the hybrid model could surpass the performance of the standalone model much

ahead of time. After running the Breakout-v0 under a standalone model for 2.5+e5 (25 Million) global steps, the highest score it could achieve was a little over the range of 12, whereas the hybrid model could surpass the same level in almost half of its execution time. In continuation to this, it is reasonable to conclude that hybrid multi-task learning by having a group of different but semantically similar environments with similar tasks could reduce the impact of partial observability that restricts a DRL agent from choosing the optimal action while in a state. Due to the impacts of the positive knowledge transfer facilitated by the gradient transfers from the second environment's agents, the actor module within each worker is having a better policy to choose the optimal action while in a step. Having said this, by possessing better policy parameters actor module is in a better position to explore the environment in a much effective way and choose the optimal action in each state. This in turn is expected to improve throughout the DRL agents' execution as more positive knowledge transfer is anticipated to happen with more global steps of game play. The same kind of comparison case study could be applied to other game test pairs from the experiment. Seen in the light of these observations, it is reasonable to conclude that the hybrid multi-task learning model is able to address the objectives, it was aiming for, to a great extent. In general, the operation of a DRL agent within its environment is always governed by either exploration or exploitation. Often when an agent starts functioning within an environment, it starts with zero knowledge and explores the environment by taking random actions. Over a period of time DRL agent accumulates a reasonable amount of knowledge from the exploration process, so that for decisions related to future actions it could exploit the knowledge which is already gathered. When it comes to the hybrid multi-task learning model, there are multiple task workers running simultaneously within the

hybrid environment and share the knowledge among them. By having a global network that consolidates the knowledge from individual task workers and then shares the revised parameter list with individual workers, each of the task workers are in a better position to derive optimal policies to make their exploration as effective as possible. This effective exploration gives the benefit of speeding up the learning process of the DRL agent which indirectly helps them to optimize their performance in terms of the rewards received at each state.

Finally, we also would like to have a comparison of the proposed hybrid multi-task learning model against the three state-of-the-art techniques that were mentioned under the related work. In comparison to the hybrid multi-task model which relies on the idea of sharing the network learning parameters by a global network to individual workers, the DISTRAL model works on the idea of sharing a distilled centroid policy that would regularize the workers running in the environment. When it comes to the comparison with the IMPALA model, its design approach is having similarity to the hybrid multi-task learning model in terms of the actor-critic methodology as it follows the topology of a set of actors with either a single learner or multiple learners. Within the IMPALA model, the learner's role is to create a central policy to be shared with the actors. Along with these learners have the flexibility to communicate among themselves for sharing the gradients. In the hybrid multi-task model, workers' accumulated gradients transfer or knowledge transfer among workers always governed by the global network. The knowledge transfer happens between the task workers running within the different environments under the hybrid multi-task learning model in the form of gradient transfer. As each of the individual worker tasks is based on the A3C-based model, the gradients shared by them with the

global network position the hybrid multi-task learning model to do a positive knowledge transfer among the individual games. This will be reflected within each environment when the updated parameter list that consolidates the combined learning parameters from individual task workers is shared by the global network to each task worker. Additionally, the current implementation of the hybrid multi-task model mandates that all the workers be present on the same machine, where the IMPALA model supports distributed system-based working environment for the workers. The PopArt model is being considered as an extension of the IMPALA model itself and designed to address key issues such as distraction dilemma and thereby stabilize the process of multi-task learning. When it comes to the adopted learning methodology, the hybrid model follows an actor-critic-based A3C algorithm-oriented technique whereas all the other models are based on an actor-learner method. Likewise, the DISTRAL model, a hybrid multi-task model is designed to function under a single-machine environment, and models such as IMPALA and PopArt can support both single machines as well as distributed machines-based environments. The hybrid multi-task model follows a gradient-based knowledge sharing by the workers with the global network. In the case of IMPALA and PopArt, actors share the trajectories of experience (3-member tuple of state, action, rewards) with the global network for sharing the knowledge. Knowledge sharing within the DISTAL is based on sharing the individual policies by the actors with the learner to derive the centroid policy [72]. The following chapter concludes the thesis and presents future work.

Table 5.1. Comparison of the Hybrid Model with State-of-the-art Solutions.

Feature Name	Hybrid Model	DISTRAL	IMPALA	PopArt
Model	Multi-agent RL	Single-agent RL	Single-agent RL	Single-agent RL
Learning Methodology	Actor-Critic (A3C)	Actor-Learner	Actor-Learner	Actor-Learner
Operating Mode	Single Machine	Single Machine	Single Machine/ Scalable to multiple machines	Single Machine/ Scalable to multiple machines
Method of sharing Learning Parameter	Share gradients to the global network	Share individual policies of actors' with learner	Share experience trajectories with learner	Share experience trajectories with learner
Multi-task Learning Approach	Multi-threaded A3C	Centroid policy	Centroid policy	Centroid policy
Knowledge Transfer Method	Parameter sharing by a global network	Regularization by learner	Policy sharing by the learner	Policy sharing by the learner

Chapter 6

Conclusion and Future Work

This thesis presented the design and evaluation of a hybrid multi-task learning model. The design of the proposed hybrid model emphasizes the applicability of the actor-critic methodology and then attempted to leverage its parallel multi-task learning capabilities by using A3C across multi- gaming(hybrid) environments. A hybrid model-based multi-task learning approach facilitates the optimization of the DRL agent's performance. The subsequent level of optimization achieved by the DRL agent relies on the hybrid model's multi-task learning capability which in turn guides the actor module to choose the best possible action at every state (as a policy π). Following this critic calculates that particular state's value which further leads to the advantage of that action.

The implementation majorly covers the information of the multi-task worker agent model and the usage of multi-task worker agents for achieving multi-task learning within the Atari 2600 based gaming environments. This chapter also outlines the information on the testing carried out under a cloud-based environment namely Paperspace, which is having GPU with CUDA support. Along with this, this section also has outlined the software packages that were selected for the construction of the model. This includes details related to tools and libraries such as PyCharm, TensorFlow, and OpenAI Gym package.

Evaluation and related results were obtained for the implementation of the presented hybrid multi-task learning model under the Atari 2600 gaming environment. The model was tested extensively with two pairs of Atari 2600 games, with each of the pair having games with a high level of semantic similarity to measure the impact of the hybrid multi-task learning

model in optimizing the performance of the DRL agent. Similarly, experiments were also conducted to measure the impact of negative knowledge transfer on the model by running the experiments on pairs of games having a high level of dissimilarity. Additionally, the model was also used to test the impact on performance when a greater number of Atari 2600 games with a high level of semantic similarity are executed together with the hybrid multi-task learning model. For all the experiments conducted, related test results are generated to reflect the impact on the DRL agent's performance. The current implementation of the hybrid multi-task learning model could achieve performance optimization with hybrid environments that are having two games, but the model's performance was found to be suboptimal when the number of games increased beyond two. This limitation demands the hybrid model to have an additional design component to mitigate the impacts of negative knowledge transfer.

For future work,

- Conduct the experiments of the hybrid multi-task model with more complex gaming environments having a higher number of worker threads under GPU cloud server-based machine environment to draw strong conclusions on hybrid multi-task learning.
- Extend the validation of the hybrid multi-task learning model with real-time application-oriented environments and investigate the changes needed to achieve performance optimization.
- As part of the test data analysis, come up with a mathematical based equation to measure the level of performance optimization achieved.

- Investigate the steps to mitigate the impacts of negative knowledge transfer and catastrophic forgetting in deep reinforcement multi-task learning.

Bibliography

[1] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in neural information processing systems*, 1996, pp. 1038-1044.

[2] C. J. Watkins, and P. Dayan, "Q-learning," *Machine learning,* vol. 8, no. 3-4, pp. 279-292, 1992.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, "Human-level control through deep reinforcement learning," *nature,* vol. 518, pp. 529-533, 2015.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928-1937.

[5] A. Mujika, "Multi-task learning with deep model based reinforcement learning," *arXiv preprint arXiv:1611.01457,* 2016.

[6] R. Glatt and A. H. R. Costa, "Improving deep reinforcement learning with knowledge transfer," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[7] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley \& Sons, 2014.

[8] L. P. Kaelbling, M. L. Littman and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence,* vol. 101, no. 1-2, pp. 99-134, 1998.

[9] R. S. Sutton and A. G. Barto, , Reinforcement learning: an introduction, Cambridge, MA: MIT Press, 1998.

[10] A. Waibel, "Modular construction of time-delay neural networks for speech recognition," *Neural computation,* vol. 1, no. 1, pp. 39-46, 1989.

[11] A. Maurer, M. Pontil and B. Romera-Paredes, "The benefit of multitask representation learning," *Journal of Machine Learning Research,* vol. 17, no. 81, pp. 1-32, 2016.

[12] R. Caruana, "Multitask learning," *Machine learning,* vol. 28, no. 1, pp. 41-75, 1997.

[13] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242-264.

[14] N. Vithayathil Varghese and Q. H. Mahmoud, "A survey of multi-task deep reinforcement learning," *Electronics,* vol. 9, no. 9, p. 1363, 2020.

[15] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering,* vol. 22, no. 10, pp. 1345-1359, 2009.

[16] G. Boutsioukis, I. Partalas and I. Vlahavas, "Transfer learning in multi-agent reinforcement learning domains," in *European Workshop on Reinforcement Learning*, Springer, 2011, pp. 249-260.

[17] G. Weiss, Multiagent systems: a modern approach to distributed artificial intelligence, MIT press, 1999.

[18] K. Weiss, T. M. Khoshgoftaar and D. Wang, "A survey of transfer learning," *Journal of Big data,* vol. 3, no. 1, pp. 1-40, 2016.

[19] D. Borsa, T. Graepel and J. Shawe-Taylor, "Learning shared representations in multi-task reinforcement learning," *arXiv preprint arXiv:1603.02041,* 2016.

[20] Y. Bengio, Learning deep architectures for AI, Now Publishers Inc, 2009.

[21] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam and M. Lanctot, "Mastering the game of Go with deep neural networks and tree search," *nature,* vol. 7587, pp. 484-489, 2016.

[22] R. S. Sutton, and A. G. Barto, Reinforcement learning: An introduction}, MIT press, 2018.

[23] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952,* 2015.

[24] J. Gideon, S. Khorram, Z. Aldeneh, D. Dimitriadis and E. M. Provost, "Progressive neural networks for transfer learning in emotion recognition," *arXiv preprint arXiv:1706.03256,* 2017.

[25] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671,* 2016.

[26] M. E. Taylor and P. Stone, "An introduction to intertask transfer for reinforcement learning," *Ai Magazine,* vol. 32, no. 1, pp. 15-15, 2011.

[27] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734,* 2017.

[28] S. Imai, S. Kawai and H. Nobuhara, "Stepwise pathnet: a layer-by-layer knowledge-selection-based transfer learning algorithm," *Scientific Reports,* vol. 140, no. 1, pp. 1-14, 2020.

[29] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295,* 2015.

[30] C. Buciluǎ, R. Caruana and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535-541.

[31] G. Hinton, O. Vinyals and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531,* 2015.

[32] W. M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz and M. Jaderberg, "Distilling policy distillation," in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 1331-1340.

[33] Z. Gu, Z. Jia and H. Choset, "Adversary a3c for robust reinforcement learning," *arXiv preprint arXiv:1912.00330,* 2019.

[34] E. Parisotto, J. L. Ba and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," *arXiv preprint arXiv:1511.06342,* 2015.

[35] M. S. Akhtar, D. S. Chauhan and A. Ekbal, "A Deep Multi-task Contextual Attention Framework for Multi-modal Affect Analysis," *ACM Transactions on Knowledge Discovery from Data (TKDD),* vol. 14, no. 3, pp. 1-27, 2020.

[36] M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research,* vol. 47, pp. 253-279, 2013.

[37] Y. Wang, J. Stokes and M. Marinescu, "Actor Critic Deep Reinforcement Learning for Neural Malware Control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020.

[38] J. Zou, T. Hao, C. Yu and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers,* 2020.

[39] A. Lazaric and M. Ghavamzadeh, "Bayesian Multitask Reinforcement Learning," in *Proceedings of International Conference on Machine Learning*, 2010.

[40] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *European workshop on reinforcement learning*, Springer, 2011, pp. 273-284.

[41] Z. Yang, K. E. Merrick, H. A. Abbass and L. Jin, "Multi-Task Deep Reinforcement Learning for Continuous Action Control.," in *IJCAI*, vol. 17, 2017, pp. 3301-3307.

[42] P. Dewangan, S. Phaniteja, K. M. Krishna and A. Sarkar, "Digrad: Multi-task reinforcement learning with shared actions," *arXiv preprint arXiv:1802.10463,* 2018.

[43] S. Omidshafiei, J. Pazis, C. Amato, J. P. How and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," *arXiv preprint arXiv:1703.06182,* 2017.

[44] S. V. Macua, A. Tukiainen, D. G.-O. Hernandez, D. Baldazo, E. M. de Cote and S. Zazo, "Diff-dac: Distributed actor-critic for average multitask deep reinforcement learning," *arXiv preprint arXiv:1710.10363,* 2017.

[45] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4496-4506.

[46] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley and I. Dunning, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561,* 2018.

[47] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt and H. van Hasselt, "Multi-task deep reinforcement learning with popart," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. Multi-task deep reinforcement learning with popart.

[48] D. Adams, D.-H. Oh, D.-W. Kim, C.-H. Lee and M. Oh, "Deep reinforcement learning optimization framework for a power generation plant considering

performance and environmental issues," *Journal of Cleaner Production,* vol. 291, p. 125915, 2021.

[49] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher and E. Hachem, "Direct shape optimization through deep reinforcement learning," *Journal of Computational Physics,* vol. 428, p. 110080, 2021.

[50] B. Hirchoua, B. Ouhbi and B. Frikh, "Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy," *Expert Systems with Applications,* vol. 170, p. 114553, 2021.

[51] Y. Du, F. Li, J. Munk, K. Kurte, O. Kotevska, K. Amasyali and H. Zandi, "Multi-task deep reinforcement learning for intelligent multi-zone residential HVAC control," *Electric Power Systems Research,* vol. 192, p. 106959, 2021.

[52] M. Andalibi, P. Setoodeh, A. Mansourieh and M. H. Asemani, "Multi-task Deep Reinforcement Learning: a Combination of Rainbow and DisTraL," in *2020 6th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, IEEE, 2020, pp. 1--6.

[53] Y. Fu, Z. Yu, Y. Zhang and Y. Lin, "Auto-Agent-Distiller: Towards Efficient Deep Reinforcement Learning Agents via Neural Architecture Search," *arXiv preprint arXiv:2012.13091,* 2020.

[54] T. T. Nguyen, N. D. Nguyen and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics,* 2020.

[55] D. S. Chaplot, L. Lee, R. Salakhutdinov, D. Parikh and D. Batra, "Embodied Multimodal Multitask Learning," *arXiv preprint arXiv:1902.01385,* 2019.

[56] T.-L. Vuong, D.-V. Nguyen, T.-L. Nguyen, C.-M. Bui, H.-D. Kieu, V.-C. Ta, Q.-L. Tran and T.-H. Le, "Sharing experience in multitask reinforcement learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, AAAI Press, 2019, pp. 3642-3648.

[57] I. Grondman, L. Busoniu, G. A. Lopes and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews,* vol. 42, no. 6, pp. 1291-1307, 2012.

[58] L. Weng, "https://lilianweng.github.io/," 18 April 2018. [Online]. Available: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#actor-critic. [Accessed 28 12 2020].

[59] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville and Y. Bengio, "An actor-critic algorithm for sequence prediction," *arXiv preprint arXiv:1607.07086,* 2016.

[60] T. Chesebro and A. Kamko, "Learning Atari: An Exploration of the A3C Reinforcement Learning Method," 15 December 2016. [Online]. Available: https://bcourses.berkeley.edu/files/70573736/download?download_frd=1. [Accessed 17 October 2020].

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540,* 2016.

[62] l. programmer, "Deep reinforcement learning in python," 24 August 2020. [Online]. Available: https://github.com/lazyprogrammer. [Accessed 21 October 2020].

[63] A. M. Taqi, A. Awad, F. Al-Azzo and M. Milanova, "The impact of multi-optimizers and data augmentation on TensorFlow convolutional neural network performance," in *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, IEEE, 2018, pp. 140-145.

[64] P. Palanisamy, Hands-On Intelligent Agents with OpenAI Gym: Your guide to developing AI agents using deep reinforcement learning, Packt Publishing Ltd, 2018.

[65] S. Dutta, Reinforcement Learning with TensorFlow: A beginner's guide to designing self-learning systems with TensorFlow and OpenAI Gym, Packt Publishing Ltd, 2018.

[66] P. Goldsborough, "A tour of tensorflow," *arXiv preprint arXiv:1610.01178,* 2016.

[67] N. Shukla and K. Fricklas, Machine learning with TensorFlow, Manning Greenwich, 2018.

[68] Q. N. Islam, Mastering PyCharm, Packt Publishing Ltd, 2015.

[69] N. D. Nguyen, T. T. Nguyen and S. Nahavandi, "A Visual Communication Map for Multi-Agent Deep Reinforcement Learning," *arXiv preprint arXiv:2002.11882,* 2020.

[70] Z. Luo, A. Small,, L. Dugan and S. Lane, "Cloud Chaser: real time deep learning computer vision on low computing power devices," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041, International Society for Optics and Photonics, 2019, p. 110412Q.

[71] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," in *International conference on high-performance computing*, Springer, 2007, pp. 197-208.

[72] N. V. Varghese and Q. H. Mahmoud, "A Hybrid Multi-Task Learning Approach for Optimizing Deep Reinforcement Learning Agents," *IEEE Access,* pp. 44681-44703, 2021.

Appendix

Appendix A. Source Code

Source code files related to worker agent creation, feature extraction from the game images using CNN, game worker thread training, game state handling and constants are provided below. Each of these source files are responsible for handling specific functionalities that are required during the course of the hybrid multi-task learning model's test execution by using the OpeanAI Gym library for Atari2600 games.

A1. A3C worker

The objective of this source file is to create the global network as well the worker agent threads, the spawn the same. Additionally, this source file handles the test data collection, so as to generate the final test statistics by using tensor board utility.

```
hybrid_env_movs= 0
test_stop= False

device = "/gpu:0"

if MODE_LSTM:
  global_knowledge_network = HybridA3CLSTMNN(ACTION_SIZE, -1, device)
training_worker_threads = []
learning_rate_parameter = tf.placeholder("float")
grad_applier = RMSPropApplier(learning_rate = learning_rate_parameter,
                              decay = RMSP_ALPHA,
                              momentum = 0.0,
                              epsilon = RMSP_EPSILON,
                              clip_norm = GRAD_NORM_CLIP,
                              device = device)

for thread in range (PARALLEL_THREADS_SIZE):
  training_thread = HybridA3CTrainingWorker(thread, global_network,
initial_learning_rate,
learning_rate_parameter,
grad_applier, MAX_TIME_STEP, device = device)
training_threads.append(training_thread)

# Create the tensorflow session
hybridsession =
tf.Session(config=tf.ConfigProto(log_device_placement=False,
                                        allow_soft_placement=True))

hybridstart = tf.global_variables_initializer()
```

```
hybridsess.run(hybridstart)

# Test results capture with tensorboard utility
hybrid_score_parameter = tf.placeholder(tf.int32)
tf.summary.scalar("score", hybrid_score_parameter)
hybrid_all_results = tf.summary.merge_all()
hybrid_results_record = tf.summary.FileWriter(LOG_FILE,
hybridsession.graph)

def train_function(parallel_index):
  global global_steps_sofar
  training_thread = training_worker_threads[parallel_index]
  while True:
    if stop_requested:
      break
    if global_steps_sofar> MAX_TIME_STEP_PARAMETER:
      break
    diff_global_steps= training_thread.process(hybridsession,
    hybrid_env_movs, hybrid_results_record,hybrid_all_results,
    hybrid_score_parameter)

    global_steps_sofar+= diff_global_steps

def signal_handler(signal, frame):
  global stop_testing
  stop_testing = True

train_worker_threads = []
for thread in range(PARALLEL_THREADS_SIZE):
  train_threads.append(threading.Thread(target=train_function,
args=(thread,)))

signal.signal(signal.SIGINT, signal_handler)

for thread in train_threads:
  thread.start()

for t in train_threads:
  thread.join()
```

A2. Feature extraction

This source file abstracts the functionalities needed to process the game screen images by using the convolutional neural network. References were made from source such as https://programtalk.com/python-examples/tensorflow.device/ for the feature extraction methods using the convolutional networks.

```
class HybridA3CNN(object):
  def __init__(self,
               agent_mov_count,
               worker_no,
               test_platform="/cpu:0"):
    self._agent_mov_count = agent_mov_count
    self._worker_no = worker_no
    self._test_platform = test_platform

  def loss_calculate(self, hyperentropy_param):
    with tf.device(self._test_platform):
      self.a = tf.placeholder("float", [None, self._agent_mov_count])
      self.td = tf.placeholder("float", [None])
      log_pi = tf.log(tf.clip_by_value(self.pi, 1e-20, 1.0))
      hybrid_entropy_val= -tf.reduce_sum(self.pi * log_pi,
reduction_indices=1)

      actorpolicyloss = -
tf.reduce_sum(tf.reduce_sum(tf.multiply(log_pi, self.a),
      reduction_indices=1) * self.td + hybrid_entropy_val_param*
hyperentropy_param)
      self.r = tf.placeholder("float", [None])
      criticvalueloss = 0.5 * tf.nn.l2_loss(self.r - self.v)
      self.total_loss = actorpolicyloss + criticvalueloss

  def synchcronize_process(self, src_netowrk, name=None):
    source_variables = src_netowrk.get_vars()
    destination_variables = self.get_vars()

    synchcronize_operations = []
    with tf.device(self._test_platform):
      with tf.name_scope(name, "HybridA3CNN", []) as name:
        for(source_variables, destination_variables) in
zip(source_variables, destination_variables):
          synchcronize_operation = tf.assign(destination_vaiables,
source_variables)
          synchcronize_operations.append(synchcronize_operation)

        return tf.group(*sync_ops, name=name)

  def _fc_variable(self, weight_shape):
    input_channels  = weight_shape[0]
    output_channels = weight_shape[1]
    d = 1.0 / np.sqrt(input_channels)
    bias_shape = [output_channels]
    weight = tf.Variable(tf.random_uniform(weight_shape, minval=-d,
maxval=d))
    bias   = tf.Variable(tf.random_uniform(bias_shape,   minval=-d,
maxval=d))
    return weight, bias

  def _conv_variable(self, game_img_align):
    w = game_img_align[0]
    h = game_img_align[1]
```

```
    input_channels  = game_img_align[2]
    output_channels = game_img_align[3]
    d = 1.0 / np.sqrt(input_channels * w * h)
    bias_shape = [output_channels]
    weight = tf.Variable(tf.random_uniform(game_img_align, minval=-d,
maxval=d))
    bias   = tf.Variable(tf.random_uniform(bias_shape,   minval=-d,
maxval=d))
    return weight, bias
  def _hybridcnn2d(self, x, W, stridesize):
    return tf.nn.conv2d(x, W, strides = [1, stridesize, stridesize, 1],
padding = "VALID")

class HybridA3CLSTMNN(HybridA3CNN):
  def __init__(self,
               agent_mov_count,
               worker_no
               test_platform="/cpu:0" ):
    HybridA3CNN.__init__(self, agent_mov_count, worker_no,
test_platform)

    scope_name = "net_" + str(self._worker_no)
    with tf.device(self._test_platform), tf.variable_scope(scope_name)
as scope:
      self.W_conv1, self.b_conv1 = self._conv_variable([8, 8, 4, 16])  #
stridesize=4
      self.W_conv2, self.b_conv2 = self._conv_variable([4, 4, 16, 32]) #
stridesize=2

      self.W_fc1, self.b_fc1 = self._fc_variable([2592, 256])
      self.lstm = tf.nn.rnn_cell.BasicLSTMCell(256, state_is_tuple=True)
      self.W_fc2, self.b_fc2 = self._fc_variable([256, agent_mov_count])
      self.W_fc3, self.b_fc3 = self._fc_variable([256, 1])
      self.s = tf.placeholder("float", [None, 84, 84, 4])

      hybrid_cnn_l1 = tf.nn.relu(self._hybridcnn2d(self.s,
self.W_conv1, 4) + self.b_conv1)
      hybrid_cnn_l2 = tf.nn.relu(self._hybridcnn2d(hybrid_cnn_l1,
self.W_conv2, 2) + self.b_conv2)
      hybrid_cnn_l2_flatlayer = tf.reshape(hybrid_cnn_l2, [-1, 2592])
      hybrid_fc_l1= tf.nn.relu(tf.matmul(hybrid_cnn_l2_flatlayer,
self.W_fc1) + self.b_fc1)
      hybrid_fc_l1_restruct = tf.reshape(h_fc1, [1,-1,256])
      self.step_size = tf.placeholder(tf.float32, [1])
      self.initial_lstm_state0 = tf.placeholder(tf.float32, [1, 256])
      self.initial_lstm_state1 = tf.placeholder(tf.float32, [1, 256])
      self.initial_lstm_state =
tf.nn.rnn_cell.LSTMStateTuple(self.initial_lstm_state0,
self.initial_lstm_state1)
      hybridlstm_result, self.lstm_state = tf.nn.dynamic_rnn(self.lstm,
hybrid_fc_l1_restruct,initial_state = self.initial_lstm_state,
sequence_length = self.step_size,time_major = False,
scope = scope)

      hybridlstm_result = tf.reshape(lstm_outputs, [-1,256])
      self.pi = tf.nn.softmax(tf.matmul(hybridlstm_result, self.W_fc2) +
self.b_fc2)
```

```
      v_ = tf.matmul(hybridlstm_result, self.W_fc3) + self.b_fc3
      self.v = tf.reshape( v_, [-1] )
      scope.reuse_variables()

      self.W_lstm = tf.get_variable("basic_lstm_cell/kernel")
      self.b_lstm = tf.get_variable("basic_lstm_cell/bias")
      self.reset_state()


   def reset_state(self):
    self.lstm_state_out = tf.nn.rnn_cell.LSTMStateTuple(np.zeros([1,
256]), np.zeros([1, 256])) def run_policy_and_value(self, sess, s_t):
    pi_out, v_out, self.lstm_state_out = sess.run( [self.pi, self.v,
self.lstm_state], feed_dict = {self.s : [s_t],

self.initial_lstm_state0 : self.lstm_state_out[0],

self.initial_lstm_state1 : self.lstm_state_out[1],

self.step_size : [1]} )

    return (pi_out[0], v_out[0])

  def run_policy(self, sess, s_t):
    pi_out, self.lstm_state_out = sess.run( [self.pi, self.lstm_state],
                                            feed_dict = {self.s : [s_t],

self.initial_lstm_state0 : self.lstm_state_out[0],

self.initial_lstm_state1 : self.lstm_state_out[1],
                                                                  self.step_size
: [1]} )
    return pi_out[0]

  def run_value(self, sess, s_t):
    prev_lstm_state_out = self.lstm_state_out
    v_out, _ = sess.run( [self.v, self.lstm_state],
                         feed_dict = {self.s : [s_t],
      self.initial_lstm_state0 : self.lstm_state_out[0],
      self.initial_lstm_state1 : self.lstm_state_out[1],
                                      self.step_size : [1]} )

    self.lstm_state_out = prev_lstm_state_out
    return v_out[0]

  def get_vars(self):
    return [self.W_conv1, self.b_conv1,
            self.W_conv2, self.b_conv2,
            self.W_fc1, self.b_fc1,
            self.W_lstm, self.b_lstm,
            self.W_fc2, self.b_fc2,
            self.W_fc3, self.b_fc3]
```

A3. Worker thread training

This source file handles the functionalities required to train the individual workers of the games selected for testing. Also, worker tracks the state of the game at each step, and records the score once it reaches to the terminal state. References were made from source such as https://programtalk.com/python-examples/tensorflow.device/ for the lstm based neural networks used in agent worker training .

```
class HybridA3CTrainingWorker(object):
  def __init__(self,
               worker_thread_index,
               global_knowledge_network,
               initial_learning_rate_parameter,
               learning_rate_input,
               grad_applier,
               max_global_time_step,
               device):

    self.worker_thread_index = worker_thread_index
    self.learning_rate_input = learning_rate_input_parameter
    self.max_global_stepsime_step = max_global_time_step

    if MODE_LSTM:
      self.local_network = HybridA3CLSTMNN(GAME_ACTION_SIZE,
worker_thread_index, device)


    self.local_network.loss_calulate(ENTROPY_BETA)
    with tf.device(device):
      var_refs = [v._ref() for v in self.local_network.get_vars()]
      self.gradients = tf.gradients(
        self.local_network.total_loss, var_refs,
        gate_gradients=False,
        aggregation_method=None,
        calculate_gradients_with_ops=False)

    self.apply_gradients = grad_applier.apply_gradients(
      global_knowledge_network.get_vars(),
      self.gradients )

    self.sync =
self.local_network.synchcronize_process(global_knowledge_network)
    self.game_state = HybridGameState()
    self.local_t = 0
    self.initial_learning_rate_parameter =
initial_learning_rate_parameter
    self.episode_reward = 0
    self.prev_local_t = 0

  def _anneal_learning_rate(self, global_time_step):
```

```
    learning_rate = self.initial_learning_rate_parameter *
(self.max_global_stepsime_step - global_time_step) /
self.max_global_time_step
    if learning_rate < 0.0:
      learning_rate = 0.0
    return learning_rate

  def choose_action(self, pi_values):
    return np.random.choice(range(len(pi_values)), p=pi_values)

  def _hybrid_scorecapture(self, hybridsession, hybrid_results_record,
hybrid_all_results, hybrid_score_parameter, score, hybrid_env_movs):
    summary_str = hybridsession.run(hybrid_all_results, feed_dict={
      hybrid_score_parameter: score
    })
    hybrid_results_record.add_summary(summary_str, hybrid_env_movs)
    hybrid_results_record.flush()

  def set_start_time(self, start_time):
    self.start_time = start_time

  def hybridproc(self, hybridsession, hybrid_env_movs,
hybrid_results_record, hybrid_all_results, hybrid_score_parameter):
    gamestates = []
    gameactions = []
    gamerewards= []
    gamevalues= []
    terminal_end = False
    hybridsession.run( self.sync )
    start_local_t = self.local_t

    if MODE_LSTM:
      LSTM_init = self.local_network.lstm_state_out

    for thread in range(LOCAL_STEPS_MAX):
      pi_, value_ =
self.local_network.run_policy_and_value(hybridsession,
self.game_state.s_t)
      action = self.choose_action(pi_)

      gamestates.append(self.game_state.s_t)
      gameactions.append(action)
      gamevalues.append(value_)

      if (self.worker_thread_index == 0) and (self.local_t %
LOG_INTERVAL == 0):
        print("thread "+str(self.worker_thread_index)+"\t|
pi={}".format(pi_))
        print("thread "+str(self.worker_thread_index)+"\t|
V={}".format(value_))

      self.game_state.hybridproc(action)

      gamereward= self.game_state.reward
      gameterminalstate = self.game_state.terminal

      self.episode_reward += gamereward
```

```
      gamerewards.append( np.clip(reward, -1, 1) )
      self.local_t += 1

      self.game_state.update()

      if gameterminalstate:
        gameterminal_state = True
        print("thread "+str(self.worker_thread_index)+"\t|
score={}".format(self.episode_reward))

        self._hybrid_scorecapture(hybridsession, hybrid_results_record,
hybrid_all_results, hybrid_score_parameter,
                            self.episode_reward, hybrid_env_movs)

        self.episode_reward = 0
        self.game_state.reset()
        if MODE_LSTM:
          self.local_network.reset_state()
        break

    cummulative_reward = 0.0
    if not gameterminal_state:
      cummulative_reward = self.local_network.run_value(hybridsession,
self.game_state.s_t)

    gameactions.reverse()
    gamestates.reverse()
    gamerewards.reverse()
    gamevalues.reverse()
    hybridbatch_si = []
    hybridbatch_a = []
    hybridbatch_td = []
    hybridbatch_reward = []

    for(ai, ri, si, Vi) in zip(gameactions, gamerewards, gamestates,
gamevalues):
      cummulative_reward = ri + GAMMA * cummulative_reward
      hybridtd = cummulative_reward - Vi
      hybrid_a = np.zeros([GAME_ACTION_SIZE])
      hybrid_a[ai] = 1

      hybridbatch_si.append(si)
      hybridbatch_a.append(hybrid_a)
      hybridbatch_td.append(hybridtd)
      hybridbatch_reward.append(cummulative_reward)

    cur_learning_rate = self._anneal_learning_rate(hybrid_env_movs)

    if MODE_LSTM:
      hybridbatch_si.reverse()
      hybridbatch_a.reverse()
      hybridbatch_td.reverse()
      hybridbatch_reward.reverse()

      hybridsession.run( self.apply_gradients,
                feed_dict = {
                  self.local_network.s: hybridbatch_si,
```

```
                self.local_network.a: hybridbatch_a,
                self.local_network.hybridtd: hybridbatch_td,
                self.local_network.r: hybridbatch_reward,
                self.local_network.initial_lstm_state: LSTM_init,
                self.local_network.step_size : [len(hybridbatch_a)],
                self.learning_rate_input: cur_learning_rate } )
    else:
      hybridsession.run( self.apply_gradients,
                feed_dict = {
                self.local_network.s: hybridbatch_si,
                self.local_network.a: hybridbatch_a,
                self.local_network.hybridtd: hybridbatch_td,
                self.local_network.r: hybridbatch_reward,
                self.learning_rate_input: cur_learning_rate} )

    diff_local_t = self.local_t - start_local_t
    return diff_local_t
```

A4. Game state handling

This file is used to create the instance of the game by using the gym library and then also to do the preprocessing(re-sizing and grayscale conversion) of the game images for the training.

```
class HybridGameState(object):
  def __init__(self, index,display=False, crop_screen=True,
frame_skip=4, no_op_max=30):
    self.index = index
    self._display = display
    self._crop_screen = crop_screen
    self._frame_skip = frame_skip
    if self._frame_skip < 1:
      self._frame_skip = 1
    self._no_op_max = no_op_max

    if(index == 1):
      GYM_ENV='Pong-v0'
    else:
      GYM_ENV = 'Breakout-v0'

    self.env = gym.make(GYM_ENV)

    # print "action space=", self.env.action_space
    self.reset()

  def _process_frame(self, action, reshape):
    reward = 0
    for i in range(self._frame_skip):
      observation, r, terminal, _ = self.env.step(action)
      reward += r
      if terminal:
        break
      # observation shape = (210, 160, 3)

    grayscale_observation = skimage.color.rgb2gray(observation)
```

```
    # shape (210, 160) range = [0.0, 1.0]

    if self._crop_screen:
      # resize to height=110, width=84
      resized_observation =
skimage.transform.resize(grayscale_observation, (110, 84))
      resized_observation = resized_observation.astype(np.float32)
      # crop to fit 84x84
      x_t = resized_observation[18:102,:]
    else:
      # resize to height=84, width=84
      resized_observation =
skimage.transform.resize(grayscale_observation, (84, 84))
      x_t = resized_observation.astype(np.float32)

    if reshape:
      x_t = np.reshape(x_t, (84, 84, 1))
    return reward, terminal, x_t

  def reset(self):
    self.env.reset()

    # randomize initial state
    if self._no_op_max > 0:
      no_op = np.random.randint(0, self._no_op_max + 1)
      for _ in range(no_op):
        self.env.step(0)

    _, _, x_t = self._process_frame(0, False)

    self.reward = 0
    self.terminal = False
    self.s_t = np.stack((x_t, x_t, x_t, x_t), axis = 2)

  def process(self, action):
    if self._display:
      self.env.render()
    r, t, x_t1 = self._process_frame(action, True)

    self.reward = r
    self.terminal = t
    self.s_t1 = np.append(self.s_t[:,:,1:], x_t1, axis = 2)

  def update(self):
    self.s_t = self.s_t1
```

A5. Constant values

This file is used to define the constant values such as game selection, number of global steps, neural network regularization, and discount factor setting.

```
LOCAL_STEPS_MAX = 25
```

```
RMSP_ALPHA = 0.99
RMSP_EPSILON = 0.1

#LOG_FILE = './graphSPACELSTMT5'
#LOG_FILE = './graphPGLSTMT25'
#LOG_FILE = './graphBRKLSTMT25'
#LOG_FILE = './graphSPACELSTMT5'
#LOG_FILE = './graphDEMNLSTMT25'
#LOG_FILE = './graphSPACELSTMT25'
#LOG_FILE = './graphBRKELS30MN'
LOG_FILE = './graphPHEONLS30MN'
INITIAL_ALPHA_LOW = 1e-4
INITIAL_ALPHA_HIGH = 1e-2

PARALLEL_SIZE = 8 # parallel thread size
#GYM_ENV = 'Pong-v0'
#GAME_ACTION_SIZE = 4
#GYM_ENV = 'Breakout-v0'
#GAME_ACTION_SIZE = 4
GYM_ENV = 'Phoenix-v0'
GAME_ACTION_SIZE = 8
#GYM_ENV = 'DemonAttack-v0'
#GAME_ACTION_SIZE = 6
#GYM_ENV = 'SpaceInvaders-v0'
#GAME_ACTION_SIZE = 6
INITIAL_ALPHA_LOG_RATE = 0.4226
GAMMA = 0.99
ENTROPY_BETA = 0.01
MAX_GLOBAL_TIME_STEP = 10 * 10**7
GRAD_NORM_CLIP = 40.0
MODE_LSTM = True
```